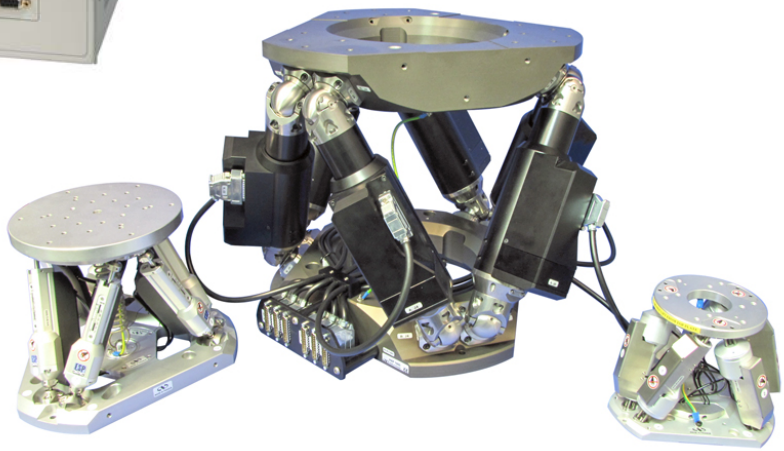
HXP

Hexapod Motion Controller



User's Manual
Software Tools
Tutorial

V3.0.x

Newport®
Experience | Solutions

Precision Motion – *Guaranteed*™

Table of Contents

User's Manual

Software Tools

Motion Tutorial

Appendix

Warranty

Newport Corporation warrants that this product will be free from defects in material and workmanship and will comply with Newport's published specifications at the time of sale for a period of one year from date of shipment. If found to be defective during the warranty period, the product will be repaired or replaced at Newport's option.

To exercise this warranty, write or call your local Newport office or representative, or contact Newport headquarters in Irvine, California. You will be given prompt assistance and return instructions. Send the product, freight prepaid, to the indicated service facility. Repairs will be made and the instrument returned freight prepaid. Repaired products are warranted for the remainder of the original warranty period or 90 days, whichever comes first.

Limitation of Warranty

The above warranties do not apply to products which have been repaired or modified without Newport's written approval, or products subjected to unusual physical, thermal or electrical stress, improper installation, misuse, abuse, accident or negligence in use, storage, transportation or handling. This warranty also does not apply to fuses, batteries, or damage from battery leakage.

THIS WARRANTY IS IN LIEU OF ALL OTHER WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR USE. NEWPORT CORPORATION SHALL NOT BE LIABLE FOR ANY INDIRECT, SPECIAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM THE PURCHASE OR USE OF ITS PRODUCTS.

EU Declaration of Conformity

NOTE

The HXP Hexapod controller is based on the same hardware as the standard XPS controller. Hence, the CE certificate of the XPS applies also to the HXP.

XPS

Newport
Experience | Solutions

Year CE mark affixed: 2015

EU Declaration of Conformity

The manufacturer:

MICRO-CONTROLE Spectra-Physics,
9, rue du bois sauvage
F-91055 Evry FRANCE

Hereby declares that the product:

Description: "XPS"
Function: Universal High-Performance Motion Controller/Driver
Type of equipment: Electrical equipment for measurement, control and laboratory use

– complies with all the relevant provisions of the Directive 2014/30/EU relating to electro-magnetic compatibility (EMC).
– complies with all the relevant provisions of the Directive 2014/35/EU relating to electrical equipment designed for use within certain voltage limits (Low Voltage)

– was designed and built in accordance with the following harmonised standards:
NF EN 61326-1:2013 « Electrical equipment for measurement, control and laboratory use – EMC requirements – Part 1: General requirements »
NF EN 55011:2010/A1:2013 Class A
NF EN 61000-3-2:2006 +A1:2009 + A2:2009 « Electromagnetic compatibility (EMC) – Part 3-2: Limits - Limits for harmonic current emissions »
CEI 61010-1:2010 « Safety requirements for electrical equipment for measurement, control and laboratory use – Part 1: General requirements »

– was designed and built in accordance with the following other standards:
NF EN 61000-4-2
NF EN 61000-4-3
NF EN 61000-4-4
NF EN 61000-4-5
NF EN 61000-4-6
NF EN 61000-4-11

Date : 26/06/2015

Dominique DEVIDAL
Quality Director

MICRO-CONTROLE Spectra-Physics
Zone Industrielle
F-45340 Beaune La Rolande, France

DC2-EN rev:A

Preface

Confidentiality & Proprietary Rights

Reservation of Title

The Newport Programs and all materials furnished or produced in connection with them (“Related Materials”) contain trade secrets of Newport and are for use only in the manner expressly permitted. Newport claims and reserves all rights and benefits afforded under law in the Programs provided by Newport Corporation.

Newport shall retain full ownership of Intellectual Property Rights in and to all development, process, align or assembly technologies developed and other derivative work that may be developed by Newport. Customer shall not challenge, or cause any third party to challenge, the rights of Newport.

Preservation of Secrecy and Confidentiality and Restrictions to Access

Customer shall protect the Newport Programs and Related Materials as trade secrets of Newport, and shall devote its best efforts to ensure that all its personnel protect the Newport Programs as trade secrets of Newport Corporation. Customer shall not at any time disclose Newport's trade secrets to any other person, firm, organization, or employee that does not need (consistent with Customer's right of use hereunder) to obtain access to the Newport Programs and Related Materials. These restrictions shall not apply to information (1) generally known to the public or obtainable from public sources; (2) readily apparent from the keyboard operations, visual display, or output reports of the Programs; (3) previously in the possession of Customer or subsequently developed or acquired without reliance on the Newport Programs; or (4) approved by Newport for release without restriction.

Sales, Tech Support & Service

North America & Asia
Newport Corporation
1791 Deere Ave.
Irvine, CA 92606, USA

Sales
Tel.: (800) 222-6440
e-mail: sales@newport.com

Technical Support
Tel.: (800) 222-6440
e-mail: tech@newport.com

Service, RMAs & Returns
Tel.: (800) 222-6440
e-mail: service@newport.com

Europe
MICRO-CONTROLE Spectra-Physics S.A.S
9, rue du Bois Sauvage
91055 Evry Cedex
France

Sales France
Tel.: +33 (0)1.60.91.68.68
e-mail: france@newport.com

Sales Germany
Tel.: +49 (0) 61 51 / 708 – 0
e-mail: germany@newport.com

Sales UK
Tel.: +44 (0)1635.521757
e-mail: uk@newport.com

Technical Support
e-mail: tech_europe@newport.com

Service & Returns
Tel.: +33 (0)2.38.40.51.55

Service Information

The user should not attempt any maintenance or service of the HXP Series Controller/Driver system beyond the procedures outlined in this manual. Any problem that cannot be resolved should be referred to Newport Corporation. When calling Newport regarding a problem, please provide the Tech Support representative with the following information:

- Your contact information.
- System serial number or original order number.
- Description of problem.
- Environment in which the system is used.
- State of the system before the problem.
- Frequency and repeatability of problem.
- Can the product continue to operate with this problem?
- Can you identify anything that may have caused the problem?

Newport Corporation RMA Procedures

Any HXP Controller/Driver being returned to Newport must have been assigned an RMA number by Newport. Assignment of the RMA requires the item serial number.

Packaging

HXP Controller/Driver being returned under an RMA must be securely packaged for shipment. If possible, re-use the original factory packaging.

User's Manual

1.0 Introduction

The HXP is a high-performance motion controller, dedicated for the use with Newport Hexapods with the ability to control an additional two single axes motion systems. One HXP is part of each Newport Hexapod system. The controller is preconfigured to the Hexapod mechanics for simplified hexapod use and integration.

The HXP is based on the same hardware as the Newport XPS Universal High-Performance Motion Controller/Driver, but uses a special firmware for Hexapod motion. The HXP has many common features with the XPS, but can control Hexapods two other single axis motion systems.

1.1 Scope of the Manual

To maximize the value of the HXP Controller/Driver system, it is important that users become thoroughly familiar with available documentation:

The **HXP Quick Start** and **HXP User's Manual** are delivered as paper copies with the controller.

Programmer's manual, TCL manual, Software Drivers manual and Stage Configuration manual are PDF files accessible from the HXP web site.

DLLs and corresponding sources are available from the controller disk in the folder Public/Drivers/DLL. DLLs can also be downloaded through FTP.

LabView VIs with examples are also available on the controller disk in the folder Public/Drivers/LabView. They can be downloaded through FTP.

To connect through FTP, please see chapter 5: "FTP (File Transfer Protocol) Connection".

The first part of this manual is the getting-started part of the system. It serves as an introduction and as a reference. It includes:

1. Introduction
2. System Overview
3. Getting Started

The second part provides a detailed description of all software tools of the HXP controller. It includes also an introduction to FTP connections and some general guidelines for troubleshooting, maintenance and service:

4. Software Tools
5. FTP (File Transfer Protocol) Connection
6. Maintenance and Service

The third part provides an exhaustive description of the HXP architecture, its features and capabilities. Different than the programmer's guide, this part is educational and is organized by features starting with the basics and getting to the more advanced features. It provides a more complete list of specifications for the different features. It includes:

7. HXP Architecture
8. Hexapod Coordinate Systems
9. Motion
10. Error Compensation
11. Event Triggers
12. Data Gathering
13. Introduction to HXP Programming

1.2 Definitions and Symbols

The following terms and symbols are used in this documentation and also appear on the HXP Series Controller/Driver where safety-related issues occur.

1.2.1 General Warning or Caution

Figure 1: General Warning or Caution Symbol.

The Exclamation Symbol in Figure 1 may appear in Warning and Caution tables in this document. This symbol designates an area where personal injury or damage to the equipment is possible.

1.2.2 Electric Shock

Figure 2: Electrical Shock Symbol.

The Electrical Shock Symbol in Figure 2 may appear on labels affixed to the HXP Series Controller/Driver. This symbol indicates a hazard arising from dangerous voltage. Any mishandling could result in damage to the equipment, personal injury, or death.

1.2.3 European Union CE Mark

Figure 3: CE Mark.

The presence of the CE Mark on Newport Corporation equipment means that it has been designed, tested and certified as complying with all applicable European Union (CE) regulations and recommendations.

1.2.4 "ON" Symbol

Figure 4: "ON" Symbol.

The "ON" Symbol in Figure 4 appears on the power switch of the HXP Series Controller/Driver. This symbol represents the "Power On" condition.

1.2.5 "OFF" Symbol

Figure 5: "OFF" Symbol.

The "Off" Symbol in Figure 5 appears on the power switch of the HXP Series Controller/Driver. This symbol represents the "Power Off" condition.

1.3 Warnings and Cautions

The following are definitions of the Warnings, Cautions and Notes that may be used in this manual to call attention to important information regarding personal safety, safety and preservation of the equipment, or important tips.

WARNING

Situation has the potential to cause bodily harm or death.

CAUTION

Situation has the potential to cause damage to property or equipment.

NOTE

Additional information the user or operator should consider.

1.4 General Warnings and Cautions

The following general safety precautions must be observed during all phases of operation of this equipment.

Failure to comply with these precautions or with specific warnings elsewhere in this manual violates safety standards of design, manufacture, and intended use of the equipment.

- Heed all warnings on the unit and in the operating instructions.
- To prevent damage to the equipment, read the instructions in this manual for selection of the proper input voltage.
- Only plug the Controller/Driver unit into a grounded power outlet.
- Assure that the equipment is properly grounded to earth ground through the grounding lead of the AC power connector.
- Route power cords and cables where they are not likely to be damaged.
- The system must be installed in such a way that power switch and power inlet remains accessible to the user.
- Disconnect or do not plug in the AC power cord in the following circumstances:
  - If the AC power cord or any other attached cables are frayed or damaged.
  - If the power plug or receptacle is damaged.
  - If the unit is exposed to rain or excessive moisture, or liquids are spilled on it.
  - If the unit has been dropped or the case is damaged.
- If the user suspects service or repair is required.
- Keep air vents free of dirt and dust.
- Keep liquids away from unit.
- Do not expose equipment to excessive moisture (>85% humidity).
- Do not operate this equipment in an explosive atmosphere.
- Disconnect power before cleaning the Controller/Driver unit. Do not use liquid or aerosol cleaners.
- Do not open the HXP Controller/Driver stand alone motion controller. There are no user-serviceable parts inside the HXP Controller/Driver.
- Return equipment to Newport Corporation for service and repair.
- Dangerous voltages associated with the 100–240 VAC power supply are present inside Controller/Driver unit. To avoid injury, do not touch exposed connections or components while power is on.
- Follow precautions for static-sensitive devices when handling electronic circuits.

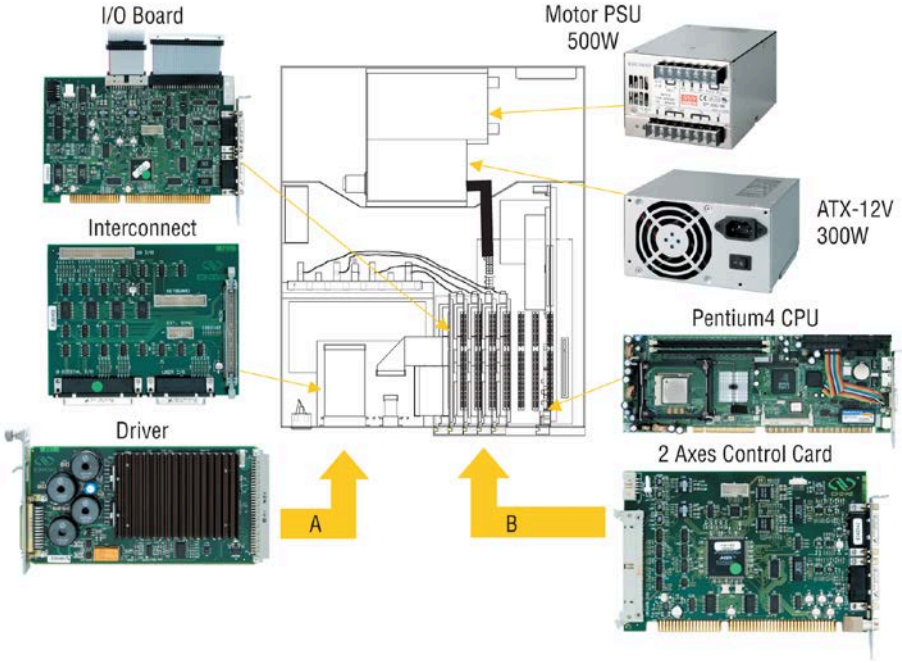
2.0 System Overview

2.1 Specifications

Number of Axes	• 6 axes of DC brush motors using internal drives • Other motion devices using external third-party drives • 1 to 2 axes of stepper, DC brush, DC brushless motors or piezo-electric stacks using internal drives
Communication Interfaces	• Internet protocol TCP/IP • One Ethernet 10/100 Base-T (RJ45 connector) with fixed IP address for local communication • One Ethernet 10/100 Base-T (RJ45 connector) for networking, dynamic addressing with DHCP and DNS • Typically 0.3 ms from sending a tell position command to receiving the answer
Firmware Features	• Powerful and intuitive, object oriented command language • Bryant angles definition (sometimes also referred to as Cardan angles or Tait-Bryan angles) • Virtual pivot motion • Tool and Work coordinate systems can be relocated by function call • Calculation of a position in the tool, work or base coordinate system to the equivalent position in the tool, work or base coordinate system. • Real time execution of custom tasks using TCL scripts • Multi-user capability • Concept of sockets for parallel processes • Data gathering at the Servo Rate, up to 1,000,000 data entries • User-defined "actions at events" monitored by the controller autonomously at the Servo Rate
Motion	• Synchronized motion of 1 to 6 Hexapod struts (real system) • Absolute move to a position in the work coordinate system (Bryant angles definition) • Incremental motion along/around the axes of the work coordinate system • Incremental motion along/around the axes of the tool coordinate system • Relocation of tool and work coordinate systems by function call • Calculation of a position in the tool, work or base coordinate system to the equivalent position in the tool, work or base coordinate system • Master slave including single master multiple slaves and custom gear ratio
Compensation	• Linear error, Backlash, positioner error mapping, hysteresis • All corrections are taken into account on the servo loop
Servo Rate	• Adjustable up to 10 kHz
Control Loop	• Open loop, PI position, PIDFF velocity, PIDFF acceleration, PIDDualFF voltage • Variable PID's (PID values depending on distance to target position) • Deadband threshold; Integration limit and integration time • Derivative cut-off filter; 2 user-defined notch filters
I/O	• 30 TTL inputs and 30 TTL outputs (open-collector) • 4 synch. analog inputs ±10 V, 14 Bit • 4 synch. uncommitted analog outputs, 16 Bit • Watchdog timer and remote interlock

Trigger In	• Hardware latch of all positions and all analog I/O's; Maximum frequency at Servo Rate • <50 ns latency on positions • CorrectorISRperiod µs time jitter on analog I/O's
Trigger Out	• One high-speed position compare output per axes that can be either configured for position synchronized pulses or for time synchronized pulses : <50 ns accuracy/latency, 2.5 MHz max. rate
Dedicated Inputs Per Axis	• RS-422 differential inputs for A, B and I, Max. 25 MHz, over-velocity and quadrature error detection • 1 Vpp analog encoder input up to x32768 interpolation used for servo; amplitude, phase and offset correction; additional 2nd hardware interpolator used for synchronization; up to x200 interpolation • Forward and reverse limit, home, error input
Dedicated Outputs Per Axis (when using external drives)	2 channel 16-bit, ±10 V D/A Drive enable, error output
Drive Capability	• Analog voltage, analog velocity, and analog acceleration (used with XPS-DRV01 and XPS-DRV03 for DC brush motor control). • Analog position (used with XPS-DRV01 for stepper motor control) • Analog position (used with external drives for example for piezo control) • Analog acceleration, sine acceleration and dual sine acceleration (used with XPS-DRV02 for brushless motors control) • Step and direction and +/- pulse mode for stepper motors (requires XPS-DRV00P and external stepper motor driver) • 500 W total available drive power
Dimensions (W x D x H)	• 19" – 4U, L: 508 mm
Weight	• 15 kg max

2.2 HXP Hardware Overview



	Hexapod Only	Hexapod and upto 2 SingleAxes
A	x6 Driver Cards	x7 or x8 Driver Cards
B	x3 CIE Boards	x4 CIE Boards

Figure 6: HXP Hardware Overview.

2.3 Front Panel Description

Figure 7: Front Panel of HXP Controller/Driver.

2.4 Rear Panel Description

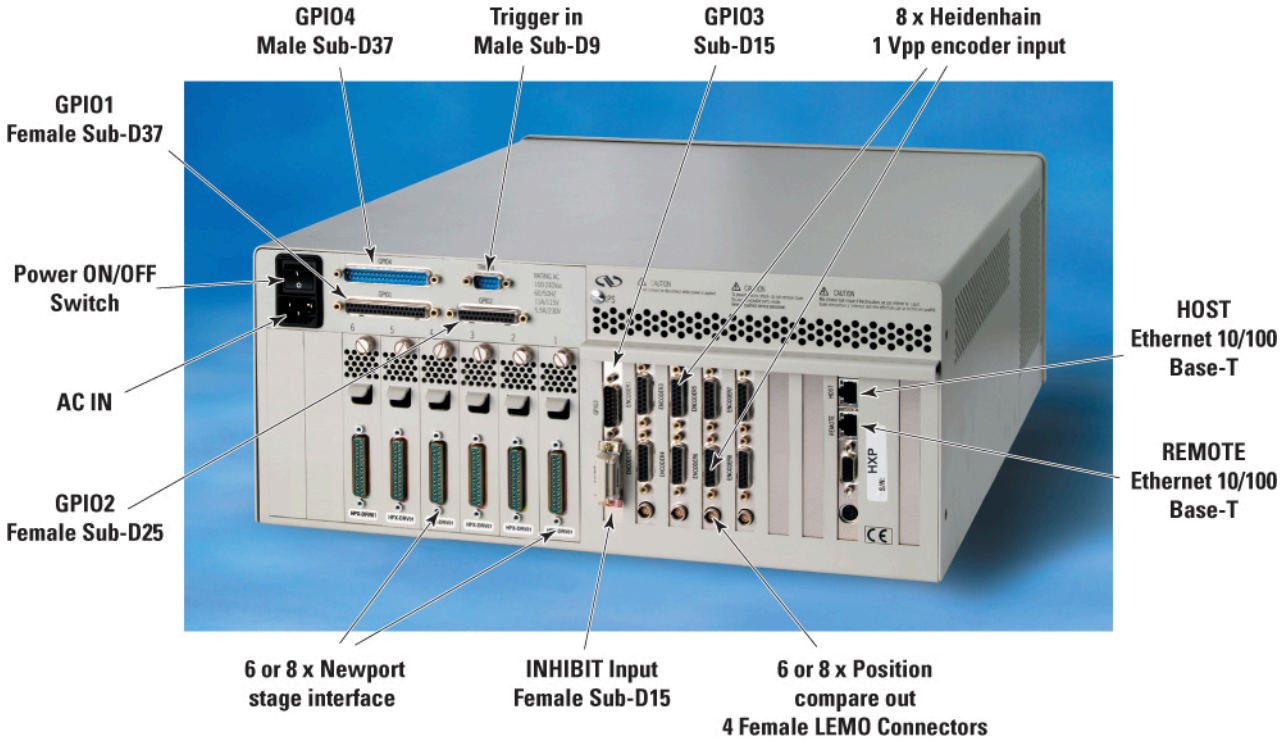


Figure 8: Rear Panel of HXP Controller/Driver.

NOTE

The Main Power ON/OFF Switch is located above the inlet for the power cord. The switch and the inlet must remain accessible to the user.

2.5 Ethernet Configuration

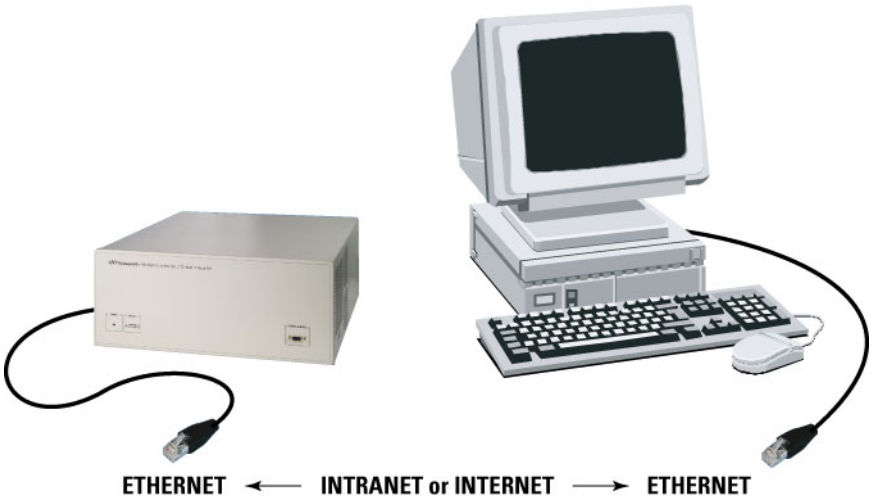


Figure 9: Ethernet Configuration.

2.5.1 Communication Protocols

The Ethernet is a local area network through which information is transferred in units known as packets. Communication protocols are necessary to dictate how these packets are sent and received. The HXP Hexapod controller supports the industry standard protocol TCP/IP.

TCP/IP is a "connection" protocol. The master must be connected to the slave in order to begin communication. Each packet sent is acknowledged when received. If no acknowledgment is received, the information is assumed lost and is resent.

2.5.2 Addressing

There are two types of addresses that define an Ethernet device. The first is the MAC address. This is a unique and permanent 6 byte number. No other device will have the same MAC address. The second type of address is the IP address. This is a 32-bit (or 4 byte) number. The IP address is constrained by each local network and must be assigned locally. Assigning an IP address to the controller can be done in different ways (see section 3.5: "Communication with the ").

2.6 Sockets, Multitasking and Multi-user Applications

Based on the TCP/IP Internet communication protocol, the HXP controller has a high number of virtual communication ports, known as sockets. To establish communication, the user must first request a socket id from the HXP controller server (listening at a defined IP number and port number). When sending a function to a socket, the controller will always reply with a completion or error message to the socket that has requested the action.

The concept and application of sockets has many advantages. First, users can split their application into different segments that run independently on different threads or even on different computers. To illustrate this, see the following example:

```
SocketID1=OpenSocket (...)
…
…
For i = 1 to nbpos
   Goal=Position (i)
   error=GroupMoveAbsolute (SocketID1, XY, goal)
   if error=OK than TakePicture
   …
Next i
…
```

```
SocketID2=OpenSocket (...)
…
…
Zerror=ReadAFSensor
error=GroupMoveRelative (SocketID2, Z, Zerror)
…
…
```

In this example, a thread on socket 1 commands an xy stage to move to certain positions to take pictures while another thread on socket 2 manages independently and concurrently an auto-focusing system. The second task could even be run on a different PC than the first task, yet be simultaneously executed within the HXP. Alternatively, if the auto-focusing system is providing an analog feedback, this task could have been also implemented as a TCL script within the HXP (see next topic)

Second, the concept of sockets has another practical advantage for many laboratory users since the use of threads allows them to share the same controller for different applications at the same time. With the HXP, it is possible that one group uses the Hexapod for one application while another group simultaneously uses other functions for a totally different application. Both applications could run completely independently from different workstations without any delays or cross-talk.

The HXP controller uses TCP/IP blocking sockets, which means that commands to the same socket are "blocked" until the HXP gives a feedback about the completion of the currently executed command (either '0' if the command has been completed successfully, or an error code in case of an error). If customers want to run several processes in parallel, users should open as many sockets as needed. Please refer to section 14.4: "Running Processes in Parallel" for further information about sockets and parallel processing.

2.7 Programming with TCL

TCL documentation is available as a PDF file and accessible from the HXP controller web site.

TCL stands for Tool Command Language and is an open-source string based command language. With only a few fundamental constructs and relatively little syntax, it is very easy to learn, yet it can be as powerful and functional as traditional C language. TCL includes many different math expressions, control structures (if, for, foreach, switch, etc.), events, lists, arrays, time and date manipulation, subroutines, string manipulation, file management and much more. TCL is used worldwide with a user base approaching one million users. It is quickly becoming a standard and critical component in thousands of corporations. Consequently TCL is field proven, very well documented and has many tutorials, applications, tools and books publicly available (www.tcl.tk).

HXP users can use TCL to write complete application code and HXP allows them to include any function to a TCL script. When developed, the TCL script can be executed in real time in the background of the motion controller processor and does not impact any processing requirements for servo updates or communication. The hardware real time multitasking operating system used on the HXP controller assures precise management of the multiple processes with the highest reliability. Multiple TCL programs run in a time-sharing mode with the same priority and will get interrupted only by the servo, communication tasks or when the maximum available time of 20 ms for each TCL program is over.

The advantage of executing application code within the controller over host run code is faster execution and better synchronization in many cases without any time taken from the communication link. The complete communication link can be reserved for time critical process interaction from or to the process or host controller.

NOTE

It is important to note that HXP provides communication requests priority over TCL script execution. When using TCL scripts for machine security or other time critical tasks, it is therefore important to limit the frequency of continuous communication requests from a host computer, which includes the HXP web-site, and to confirm the execution speed of repetitive TCL scripts.

3.0 Getting Started

3.1 Unpacking and Handling

It is recommended that the HXP Hexapod Controller be unpacked in your lab or work site rather than at the receiving dock. Unpack the system carefully; small parts and cables are included with the equipment. Inspect the box carefully for loose parts before disposing of the packaging. You are urged to save the packaging material in case you need to ship your equipment.

3.2 Inspection for Damage

The HXP controller has been carefully packaged at the factory to minimize the possibility of damage during shipping. Inspect the box for external signs of damage or mishandling. Inspect the contents for damage. If there is visible damage to the equipment upon receipt, inform the shipping company and Newport Corporation immediately.

WARNING

Do not attempt to operate this equipment if there is evidence of shipping damage or you suspect the unit is damaged. Damaged equipment may present additional hazards to you. Contact Newport technical support for advice before attempting to plug in and operate damaged equipment.

3.3 Packing List

Included with each HXP controller are the following items:

- User's Manual and Motion Tutorial.
- HXP controller.
- Cross-over cable, gray, 3 meters.
- Straight-through cable, black, 5 meters.
- Power cord.
- Rack mounting ears and handles.

If there are missing hardware or have questions about the hardware that were received, please contact Newport.

CAUTION

Before operating the HXP controller, please read chapter 1.0 very carefully.

3.4 Connecting the Hardware

If not already done, carefully unpack and visually inspect the controller and stages for any damage. Place all components on a flat and clean surface.

CAUTION

No cables should be connected to the controller at this point!

The HXP Hexapod controller is preconfigured to the supplied Hexapod hardware. Labels on the driver cards of the HXP controller refer to the different motors (struts or actuators) of the Hexapod. Similar labels are found on the motor cables of the Hexapod. It is important that the motors (struts or actuators) of the Hexapod are connected to the corresponding driver cards of the HXP controller.

Due to the high power available in the HXP controller (300 W for the CPU and 500 W for the drives), ventilation is very important.

To ensure a good level of heat dissipation these rules must be followed:

1. It is strictly forbidden to use the HXP controller without the cover properly mounted on the chassis.
2. The surrounding ventilation holes at the sides and back of the HXP rack must be free from obstructions that prevent free flow of air.

- Connect the motor cables of the Hexapod hardware to the corresponding driver cards on the rear panel of the HXP controller (see matching labels on motor cables and driver cards)
- Plug the AC line cord supplied with the HXP into the AC power receptacle on the rear panel.
- Plug the AC line cord into the AC wall-outlet. Turn the Main Power Switch to ON (located on the Rear Panel).
- The system must be installed in such a way that power switch and power connector remains accessible by the user.
- After the main power is switched on, the LED on the front panel of the HXP will turn green.
- There is an initial beep after power on and a second beep when the controller has finished booting. The time between the first and the second beep may be 1-2 minutes.
- There is also a STOP ALL button on the front panel that is used for an Emergency Stop of the system and will shut down all the motors.

CAUTION

Never connect/disconnect stages while the HXP controller is powered on.

3.5 Communication with the HXP

The HXP controller provides two Ethernet connections called HOST and REMOTE. Both connections can be used at the same time. The REMOTE connection has a fixed IP address that can not be changed by the user. The HOST has a default IP adderss that can be changed using the XPS web tools (see chapter 3.5.4). It is possible to connect to the controller in 2 ways:

1. Direct connection PC to HXP – requires use of a cross over Ethernet cable (gray).
2. Connection through a local area network – requires input from your Network Administrator and use of a straight Ethernet cable (black).

A cross-over (gray) and straight (black) Ethernet cable are supplied with the HXP.

3.5.1 Direct Connection to the HXP controller

For a direct connection between a PC and the HXP controller you need to use the gray cross-over cable.

In the following, we describe the necessary steps to connect to the HXP through the HOST connection, but connection through REMOTE is the same except for the different IP address.

For a direct connection between a PC and the HXP controller you need to use the gray cross-over cable and the HOST connector at the back of the HXP mainframe.

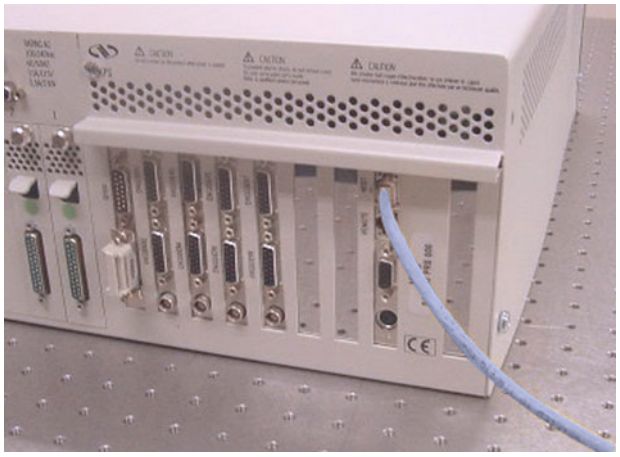
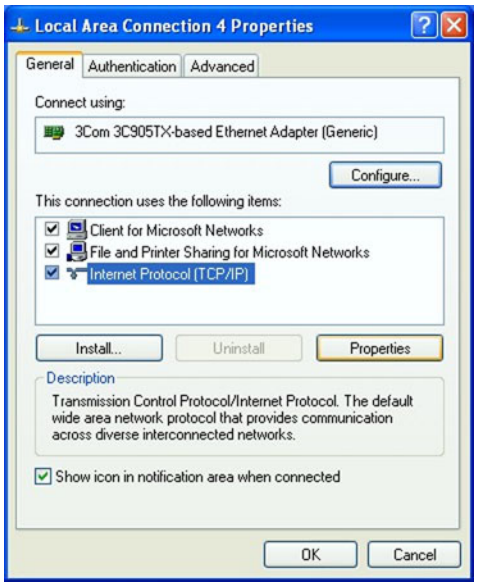


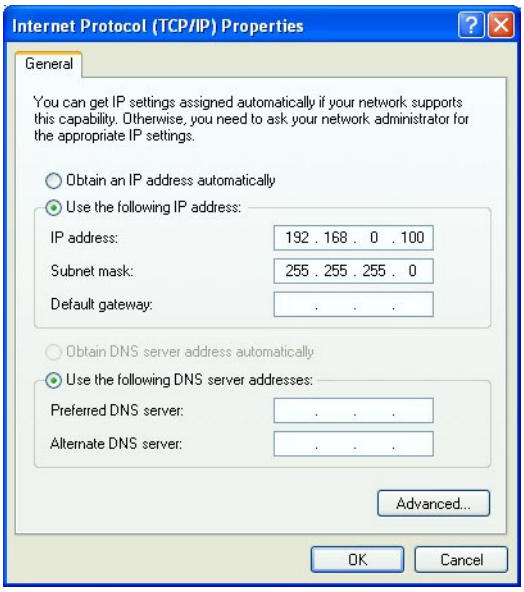
Figure 10: Direct Connection to the HXP using cross-over cable.

First, the IP address on the PC's Ethernet card has to be set to match the default factory HXP's IP address for the HOST (192.168.0.254). In case of a Windows XP operating system, this can be done as follow:

1. Start Button > Control Panel > Network Connections.
2. Right Click on Local Area Connection Icon and select Properties.



3. Highlight Internet Protocol (TCP/IP) and click on Properties.
4. Set the IP address and Subnet Mask as shown in the next window.



5. Click "OK".

NOTE

The Last number of the IP address can be set to any number between 2 to 253: 100 for example.

NOTE

When configuring the controller to be on the network, the settings for the PC's Ethernet card will have to be set back to default under "Obtain an IP address automatically".

Once the Ethernet card address is set, you are ready to connect to the HXP controller:

6. Open Internet Browser and connect to http://192.168.0.254

   Login:

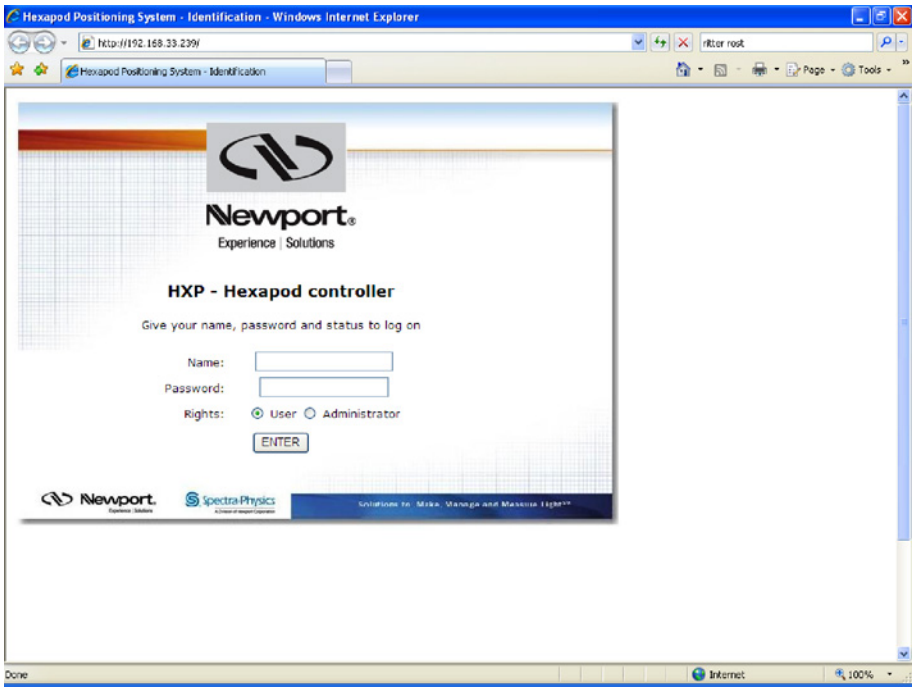
   Name: **Administrator**

   Password: **Administrator** (Please see the picture below).

   Rights: **Administrator**

NOTE

Please note that the login is case sensitive.



Your computer has now established a direct connection to the HXP.

NOTE

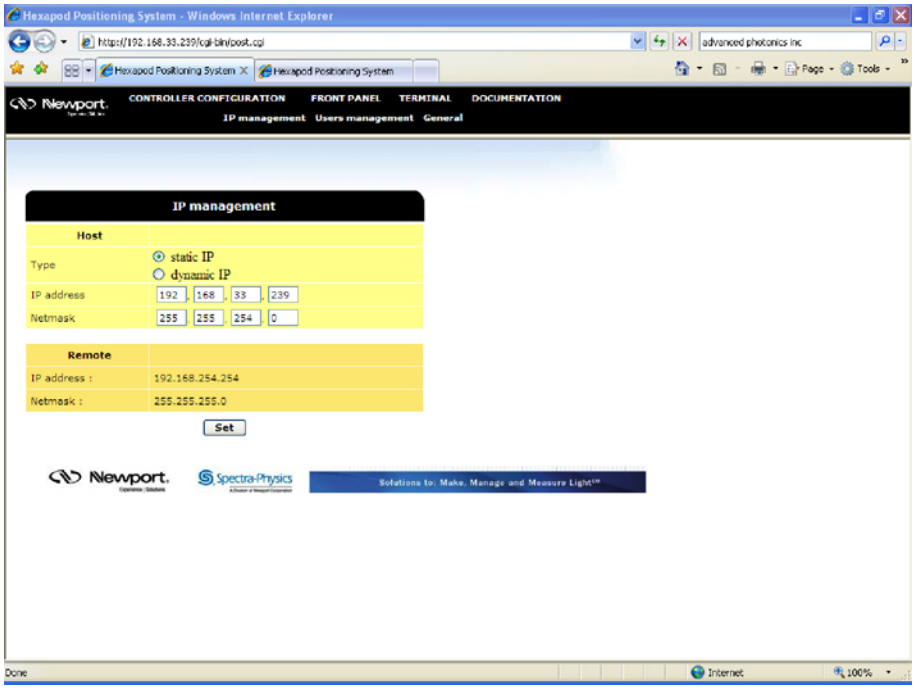
If you want to change the IP address of the HXP controller, follow the next sections, but continue using the gray, cross-over Ethernet cable.

NOTE

If you don't want to connect to the HXP controller through a network, you can skip the next sections and continue reading in section 4.0: "Software Tools".

3.5.2 Connecting the HXP to a Network Using Static IP Configuration

Once you are logged to the HXP using the previously described steps, you can change the IP configuration of the HXP controller in order to connect the HXP to a Network. Select "CONTROLLER-CONFIGURATION" of the web-site and select the sub-menu "IP-Management".



The static IP address, the subnet mask and the Gateway IP address have to be provided by your Network Administrator to avoid network conflicts. Once you have obtained these addresses you can input them in the IP configuration window as shown above. The addresses shown above are examples only.

NOTE

To avoid conflict with the REMOTE Ethernet plug, the IP address must be different from 192.168.254…

NOTE

For most networks the above setting for the Subnet Mask will work. However, for large networks (200 computers or more) the Subnet Mask address has to be verified with IT department. In most cases for larger networks Subnet Mask is set to 255.255.0.0.

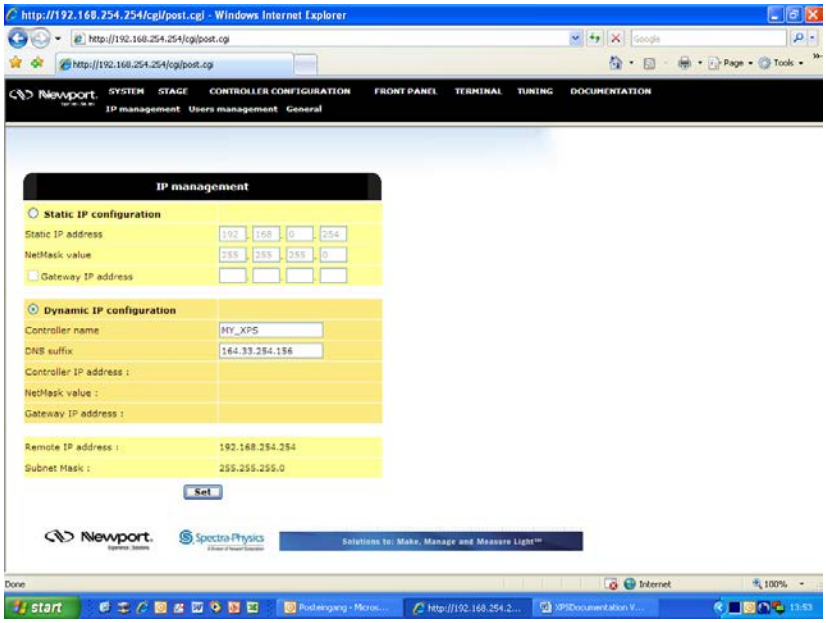
Once the appropriate addresses for the Static IP configuration are set, click on SET and switch off controller.

Connect now a CAT-5 network cable (black cable supplied with the HXP) to the HOST connector of the HXP controller and to your network.

After restarting the controller and resetting your PC's Ethernet card to the default configuration, open the Internet browser and connect using your given Static IP address.

3.5.3 Connecting the HXP to a Corporate Network Using Dynamic IP Configuration

Obtain DNS suffix from network administrator for the "DNS suffix" field.



Assign any name for the "Controller name" field.

Click on SET and switch off the controller.

Make sure that the standard CAT-5 network cable (black cable supplied with HXP) is connected to the HOST of the HXP controller and to your network.

After restarting the controller, open the internet browser and connect to the HXP using your controller name.

NOTE

Do not use Dynamic IP configuration if your DHCP server uses Windows NT 4.0 server.

3.5.4 Recovering lost IP configuration

If you want to recover a lost IP configuration, you need to connect directly the PC to the REMOTE connector at the back of the HXP mainframe with the gray cross-over cable.

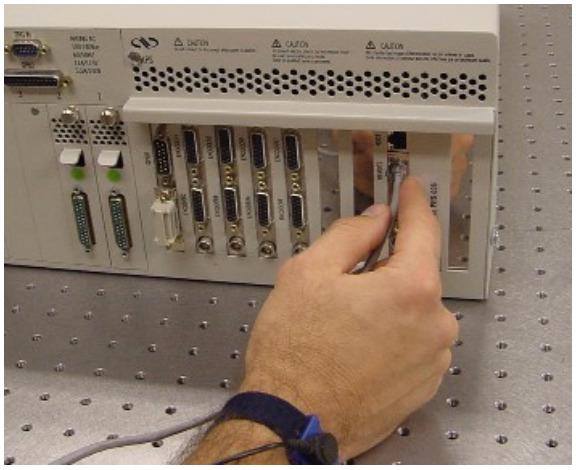
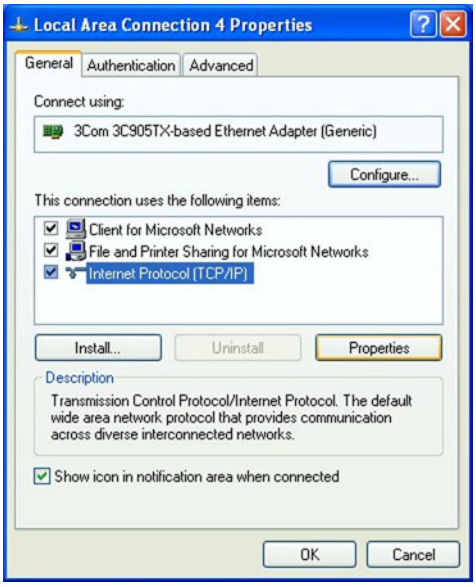


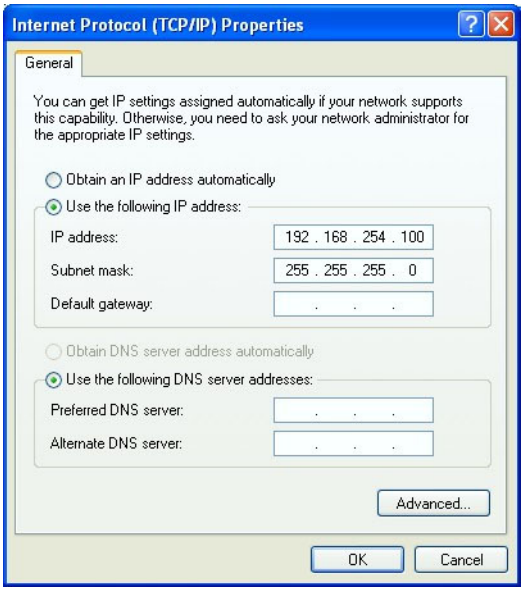
Figure 11: Direct Connection to the HXP using cross-over cable and REMOTE connector.

Set the IP address on the PC's Ethernet card to match the HXP's fixed IP address for the REMOTE plug (192.168.254.254). In case of a Windows XP operating system, this can be done as follow:

1. Start Button > Control Panel > Network Connections.
2. Right Click on Local Area Connection Icon and select Properties.



3. Highlight Internet Protocol (TCP/IP) and click on Properties.
4. Set the IP address and Subnet Mask as shown in the next window.



5. Click "OK".

NOTE

The last number of the IP address can be set to any number between 2 to 253: 100 in this example.

NOTE

When configuring the controller to be on the network, the settings for the PC's Ethernet card will have to be set back to default which is "Obtain an IP address automatically".

When the Ethernet card address is set, you can connect to the HXP controller:

6. Open Internet Browser and connect to **http://192.168.254.254**

Login:

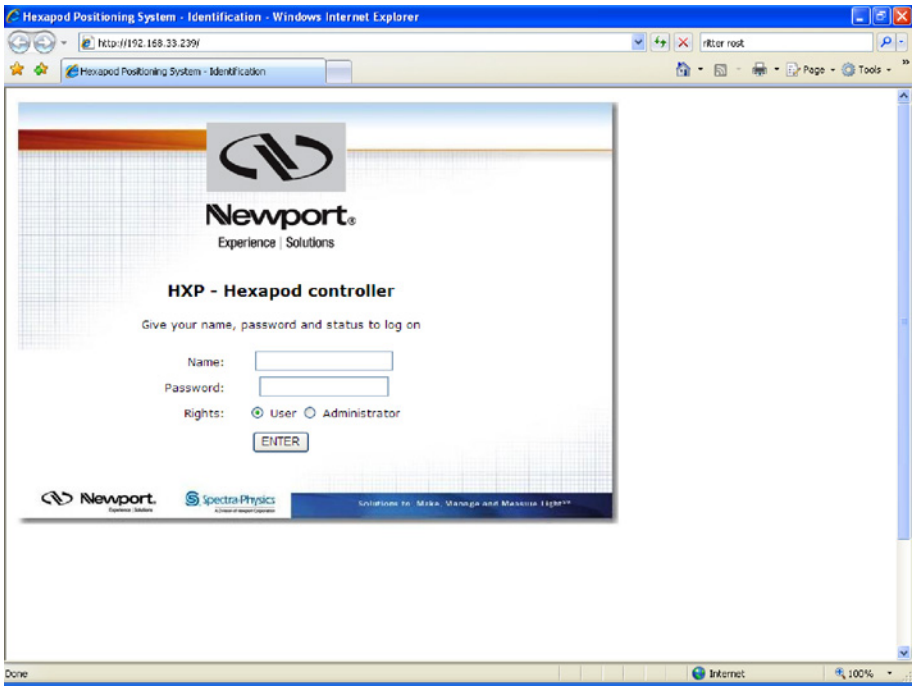
Name: **Administrator**

Password: **Administrator** (Please see the picture below).

Rights: **Administrator**

NOTE

The login name and password are case sensitive.



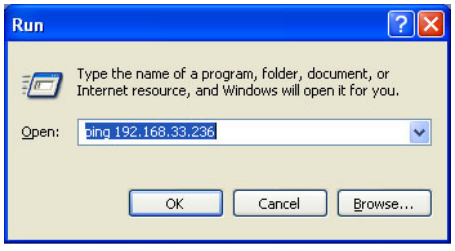
Once you are logged, you can change the IP configuration by following the steps described in section 3.5.2 or 0 depending on your configuration.

NOTE

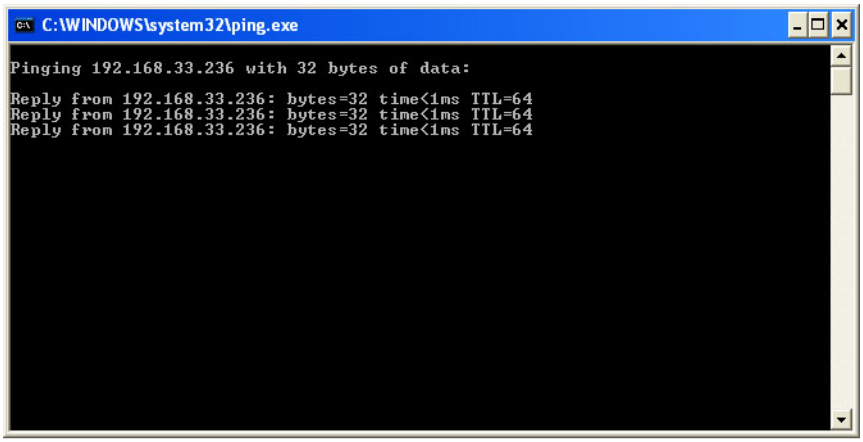
If you want to reset the IP address to the default factory setting, follows the section 3.5.2 and set the IP address to 192.168.0.254.

3.6 Testing your HXP-PC Connection

To check if the HXP is connected to the host computer, you could send a ping message from the computer to the HXP. This is done by using the menu: Start->Run->then type: ping + IP address of the HXP. See the example below for the IP address 192.168.33.236:



If the HXP is connected, it replies in the terminal window that appears when users click on the OK button:

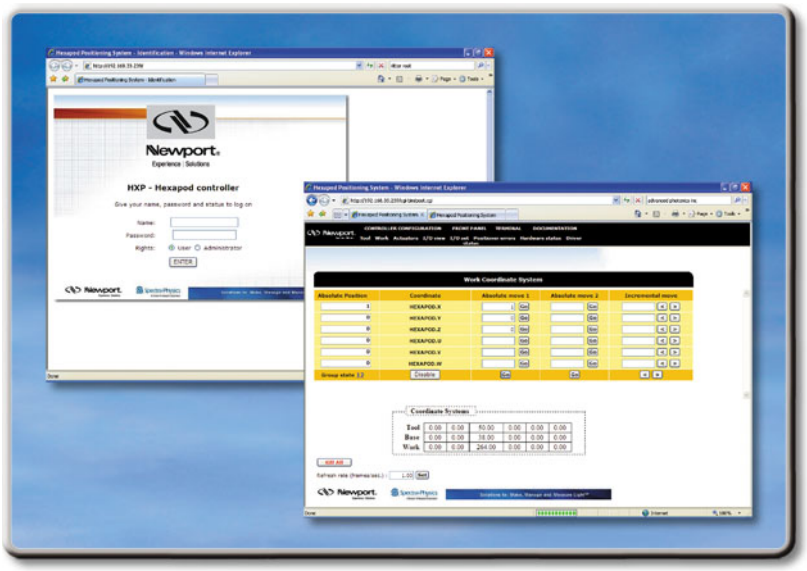


If the HXP controller is not connected, the window displays that the time delay of the request is exceeded.

3.7 System Shut-Down

To shut down the system entirely, perform the following steps:

1. Wait for the stage(s) to complete their moves and come to a stop.
2. Turn off the power, the switch is located above the power cord at the back of the controller.



Software Tools

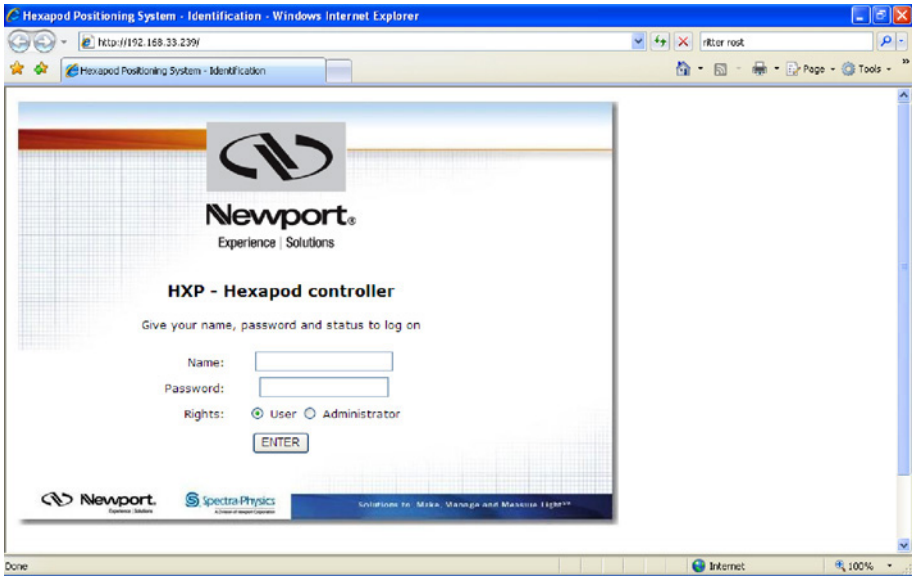
4.0 Software Tools

4.1 Software Tools Overview

The HXP software tools provide users a convenient access to the most common features and functions of the HXP controller. All software tools are implemented as a web interface. The advantage of a web interface is that it is independent from the user's operating system and doesn't require any specific software on the host PC.

There are two options to log-in to the HXP controller: As “User” or as “Administrator”. Users can log-in only with User rights. Administrators can log-in with User or with Administrator rights. When logged-in with Administrator rights, you have an extended set of tools available.

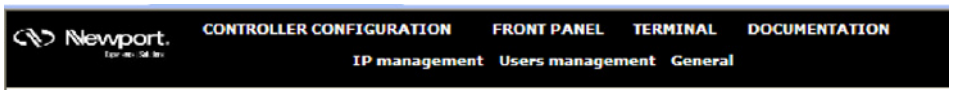
The predefined user has the log-in name **Anonymous**, Password **Anonymous**. The predefined Administrator has the log-in name **Administrator**, Password **Administrator**. Both, the Log-in name and the password are case sensitive.



The main tag is displayed across the top of the HXP Motion Controller/Driver main program window, and lists each primary interface option. Each interface option has its own pull-down menu that allows the user to select various options by clicking the mouse's left button.

On the following pages we provide a brief description of all tools.

Administrator Menus with sub-Menu for CONTROLLER CONFIGURATION



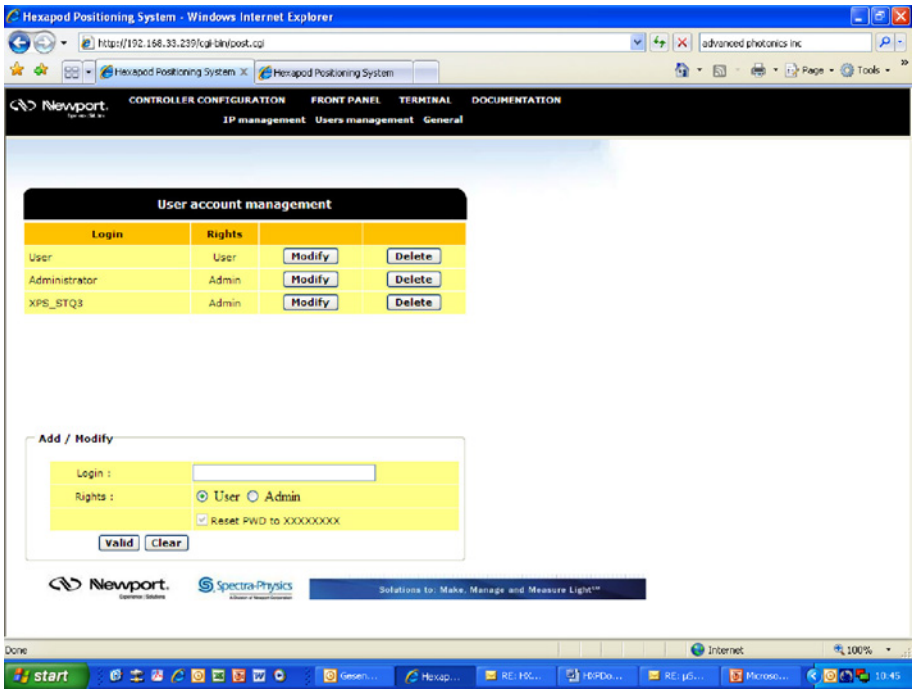
4.2 CONTROLLER CONFIGURATION – Users Management

This tool allows managing user accounts. There are two type of users defined: Administrators and Users. Administrators have configurations rights. Users have only restricted rights to use the system.

Use the following steps to create a new user:

1. Enter a new user name in the “login” field.
2. Choose the access rights: “User” or “Admin”.
3. Check the box “Reset PWD to XXXXXXXX”:

   Your password is reset to XXXXXXXX.
4. Select the “VALID” button to add the new access account.

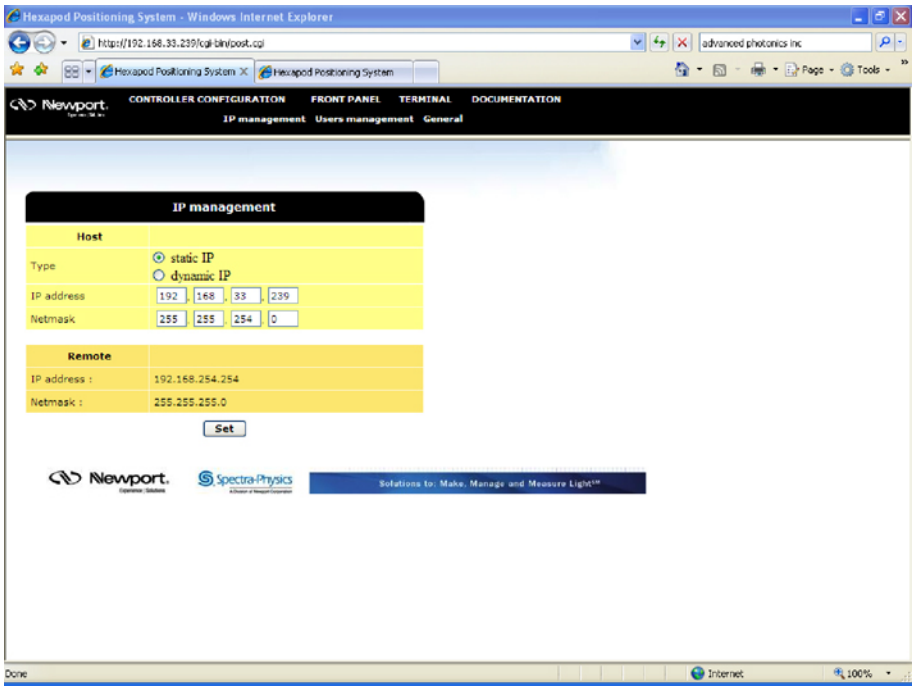


NOTE

The default password is XXXXXXXX and must be changed after the first log in.

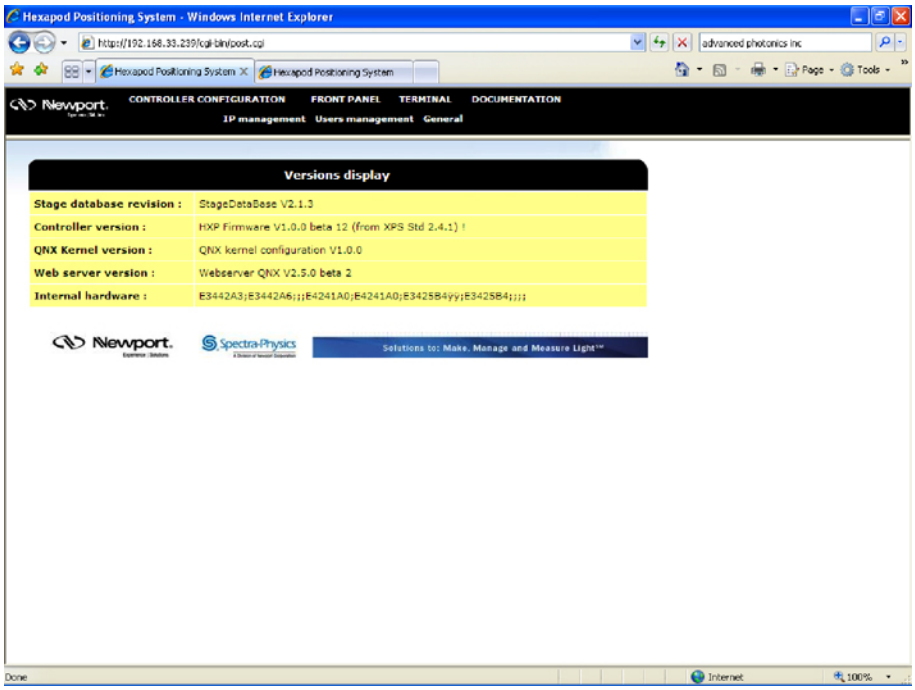
4.3 CONTROLLER CONFIGURATION – IP Management

See chapter 3.5 for details.



4.4 CONTROLLER CONFIGURATION – General

This screen provides valuable information about the firmware and the hardware of the controller. It is an important screen for troubleshooting the controller.



4.5 FRONT PANEL – Tool

The Tool page provides access to incremental moves along and around the axes of the Tool coordinate system. See chapter 9.4 for details. A couple of comments about this page:

Absolute Position refers to the position of the Tool coordinate system in the Work coordinate system, see chapter 8.0 and chapter 9.2.1 for details.

Group State refers to the state of the Hexapod group, see chapter 7.2 for details about group states. Moving the mouse over the blue number opens a window describing the group state.

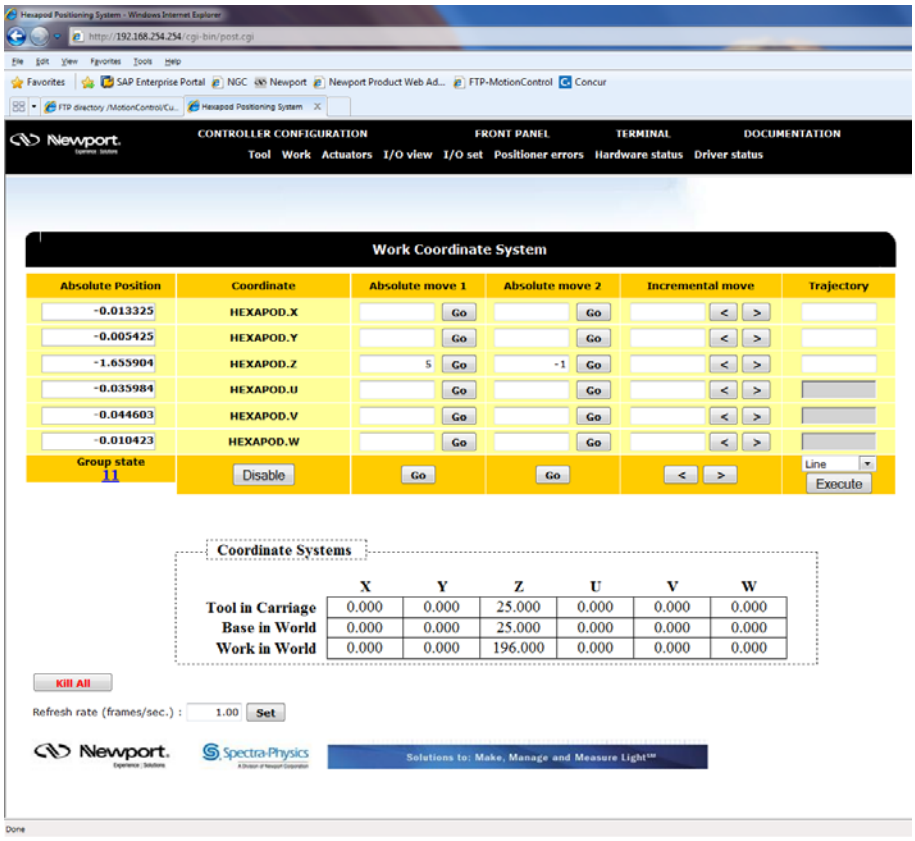
The gray field next to Group state is a dynamic action button. The meaning changes according to the group state.

Coordinate lists the coordinate names that are composed by the GroupName (default is HEXAPOD), followed by a "." and the coordinate (X, Y, Z, U, V, or W).

Under **Incremental move,** it is possible to increment individual coordinates by entering a value in one of the six fields and pressing the "<" or ">" button next to the input window. It is also possible to increment all six coordinates at the same time ("<" and ">" button in the dark orange line, but in this case entry values for all six coordinates must be provided. Zero (0) is an allowed entry value.

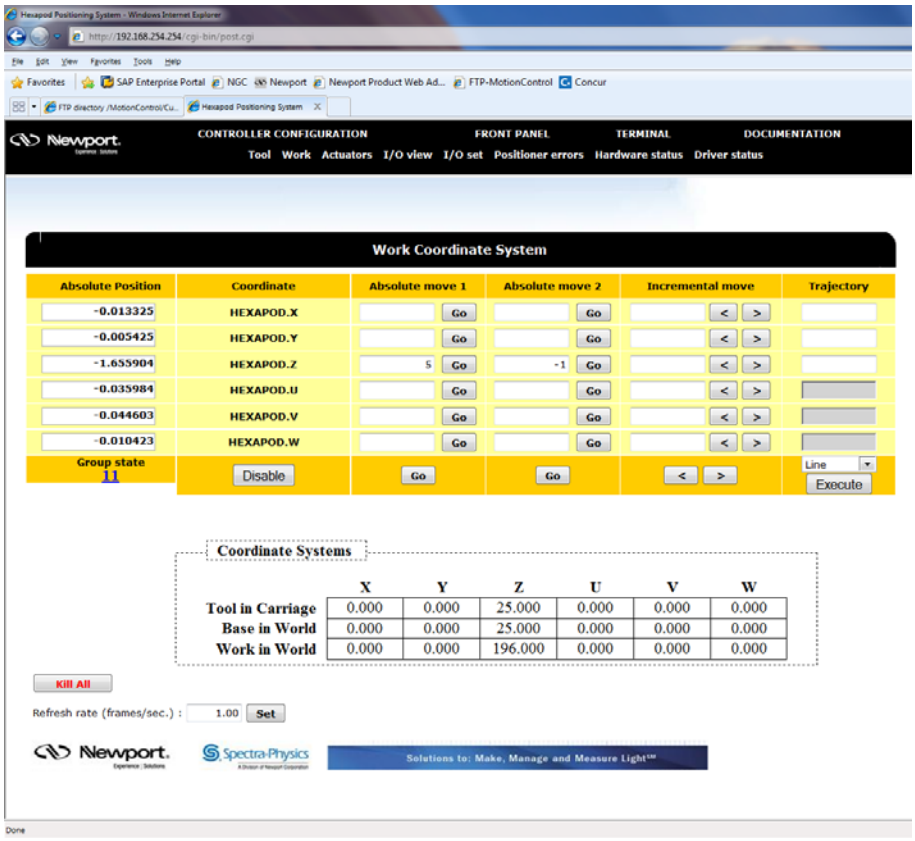
Refresh rate (frames/sec.) sets the screen update rate for the position and group state. Please note that the Coordinate Systems positions (see dashed frame) are NOT affected by this update. The Coordinate System position are only updated when reloading the whole page (F5 button with Windows Internet Explorer).

Kill All stops all motion and sets the Hexapod to the "Not Initialized" state.



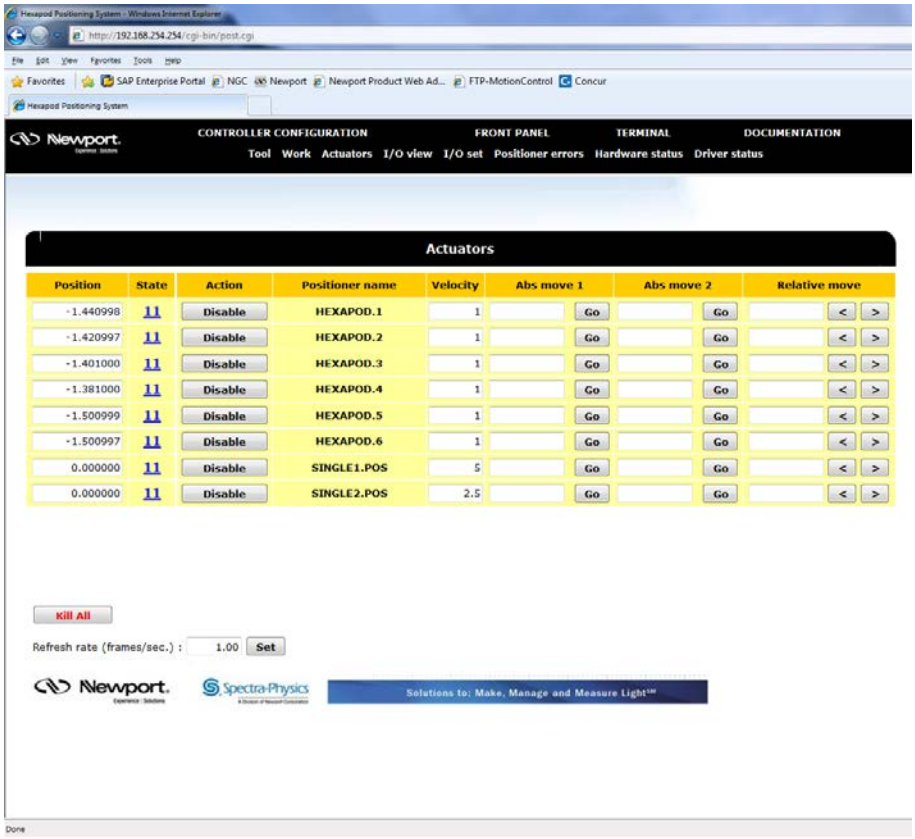
4.6 FRONT PANEL – Work

The Work page is similar to the Tool page. It provides access to absolute moves, incremental moves and RightPath Trajectory moves along and around the axes of the Work coordinate system. See chapter 9.2.1 and 9.5 for details about the definition of absolute moves and incremental moves along and around Work. See chapter 10 for details about the definition and parameter inputs for RightPath Trajectory.



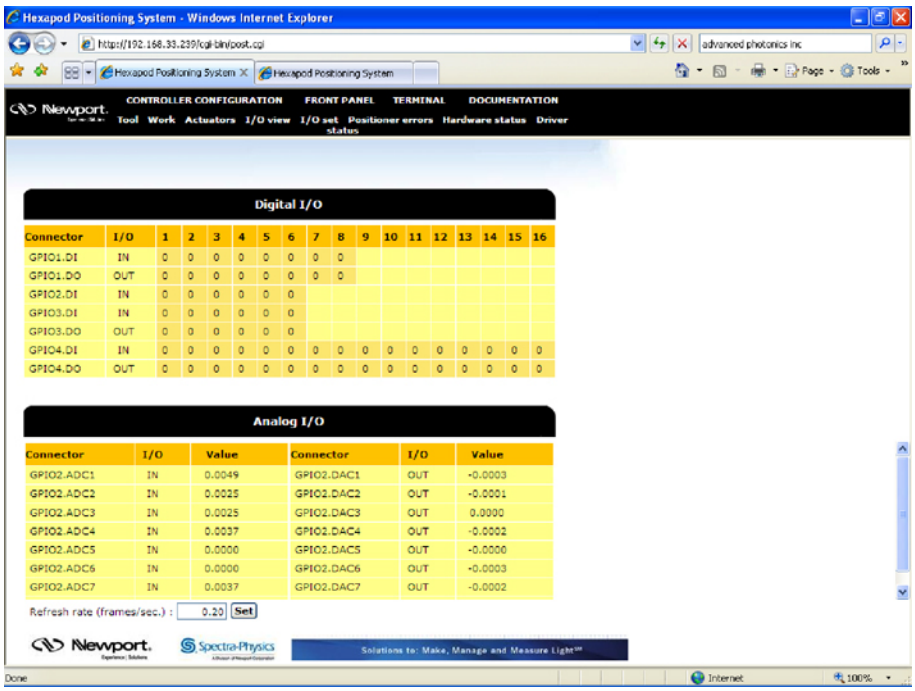
4.7 FRONT PANEL – Actuators

The Actuators page provides access to basic group functions like initialize, home, or motor disable, and executes relative and absolute moves. The Actuator page also provides a convenient review of all important group information such as group names, group states and positions. All motion groups are listed in the Actuator page. Additionally, the Actuators page lists the position of each Hexapod strut and provides access to change the position of the Hexapod struts. See chapter 9.6 for more details. There is no simple correlation between the position of the Hexapod struts and the position of the Hexapod top plate in Cartesian coordinates. It is not recommended to execute motion on individual Hexapod actuators rather this page is used for hexapod reference as well as motion of a SingleAxis group.



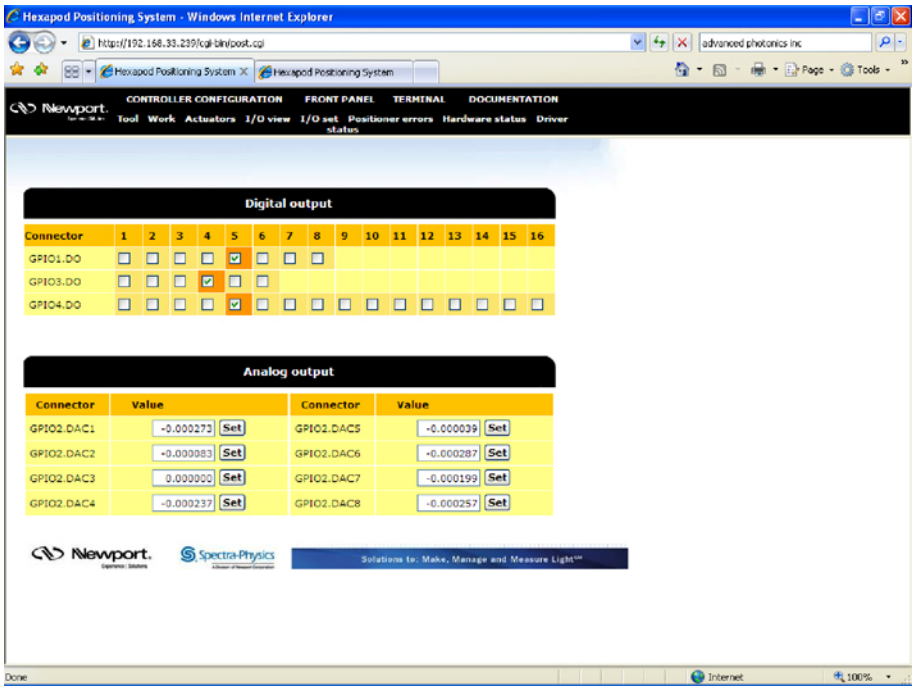
4.8 FRONT PANEL – I/O View

The I/O View page provides review of all analog and all digital I/O's of the controller. To set the outputs, use the page I/O Set.



4.9 FRONT PANEL – I/O Set

The I/O set page provides access to set the analog and digital outputs of the controller.

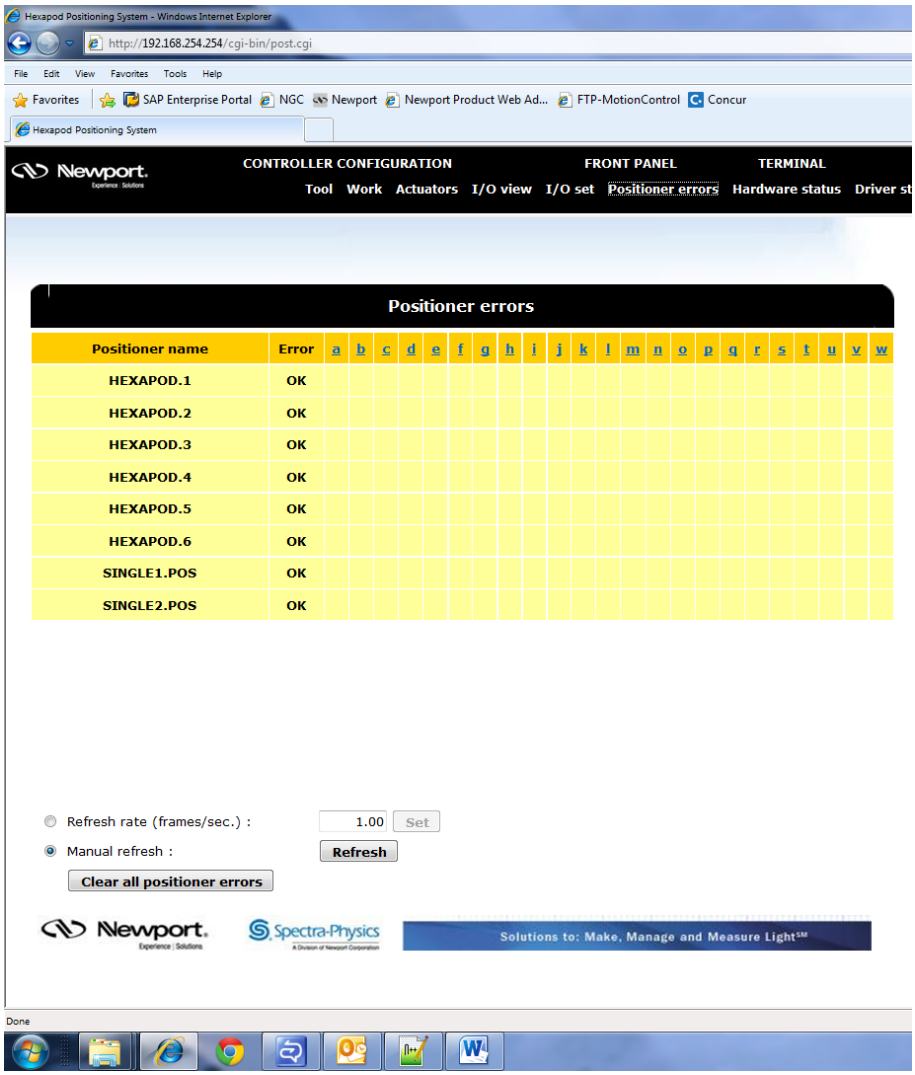


4.10 FRONT PANEL – Positioner Errors

The positioner errors page is an important page for trouble-shooting. When encountering any problems during the use of the system, you find here valuable information about the errors related to the positioners.

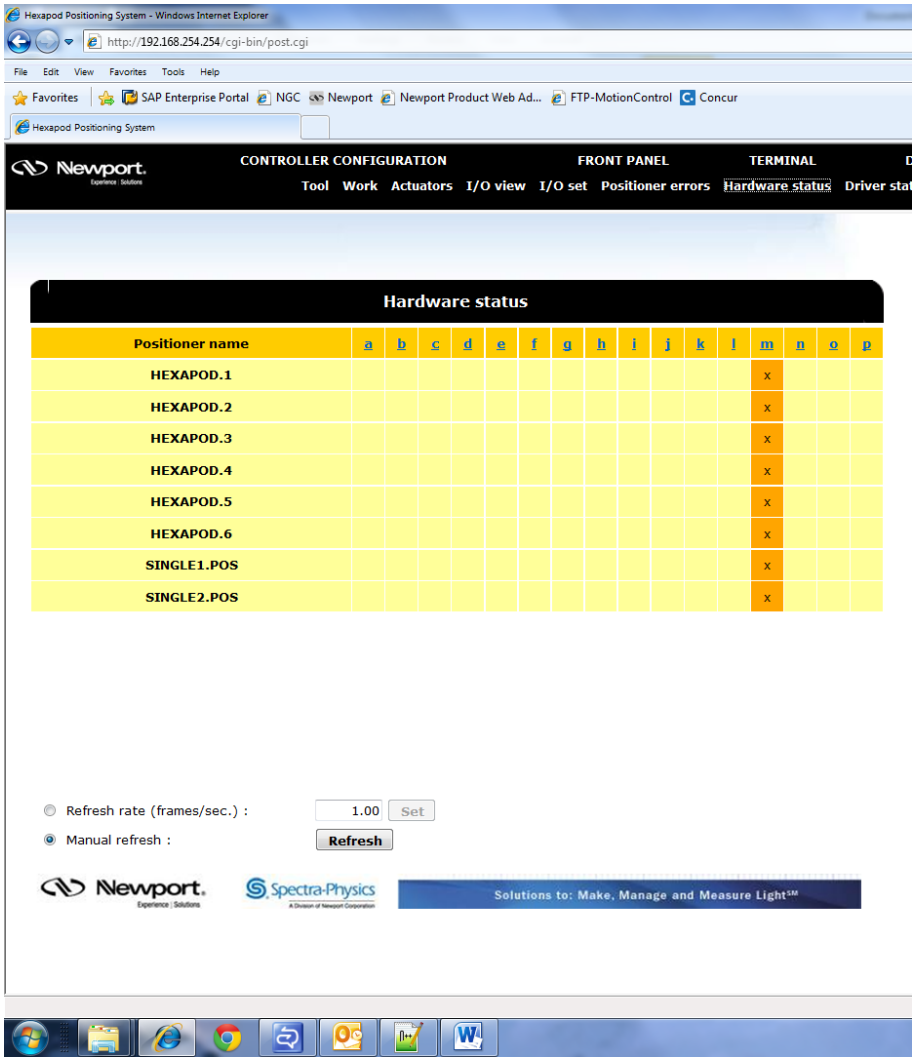
Note that all positioner errors encountered since the last “Clear all positioner errors” are displayed, even if some of the errors may not be present anymore. The button “Refresh” refreshes the error page. This means, new errors are displayed, but old errors that do not exist anymore, are not removed.

To clear the errors, use the button “Clear all positioner errors”.



4.11 FRONT PANEL – Hardware Status

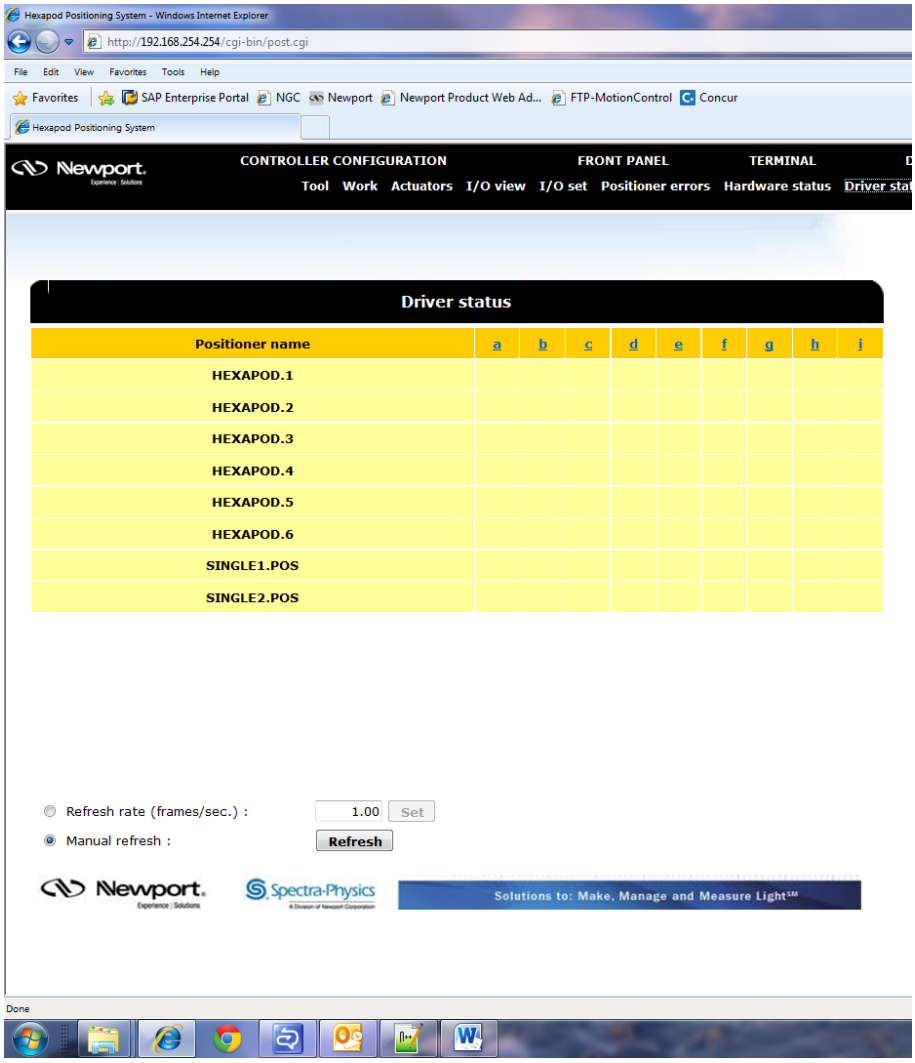
The Hardware status page is another important page for trouble-shooting. You find more valuable information related to your hardware here. Not all information is related to an error.



4.12 FRONT PANEL – Driver Status

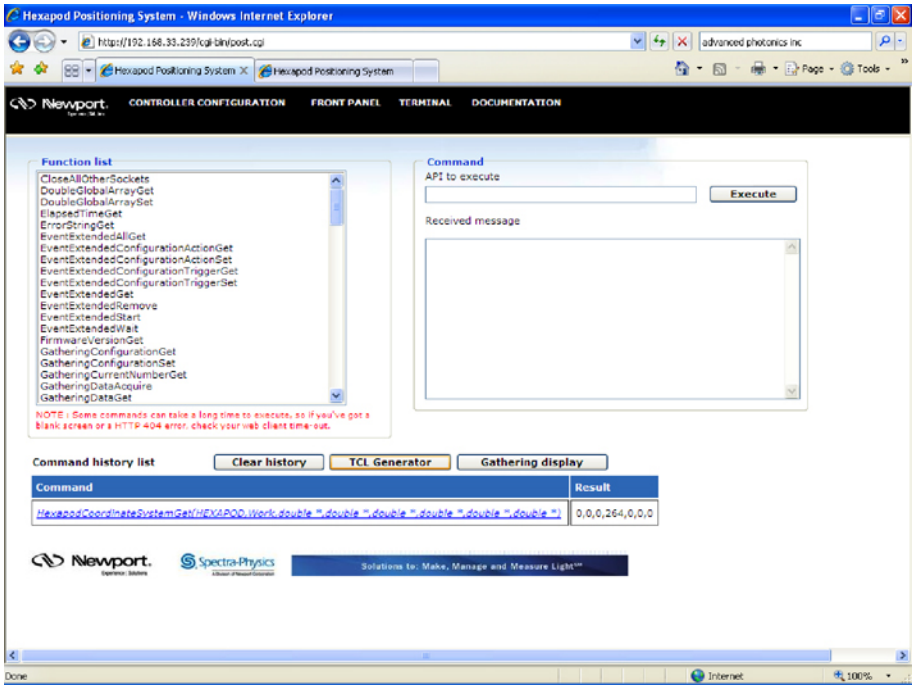
The Driver status page is another important page for trouble-shooting. Here you find valuable information related to the status of the driver. Not all information is related to an error.

The type of status information that you can get depends on the drivers used.



4.13 TERMINAL

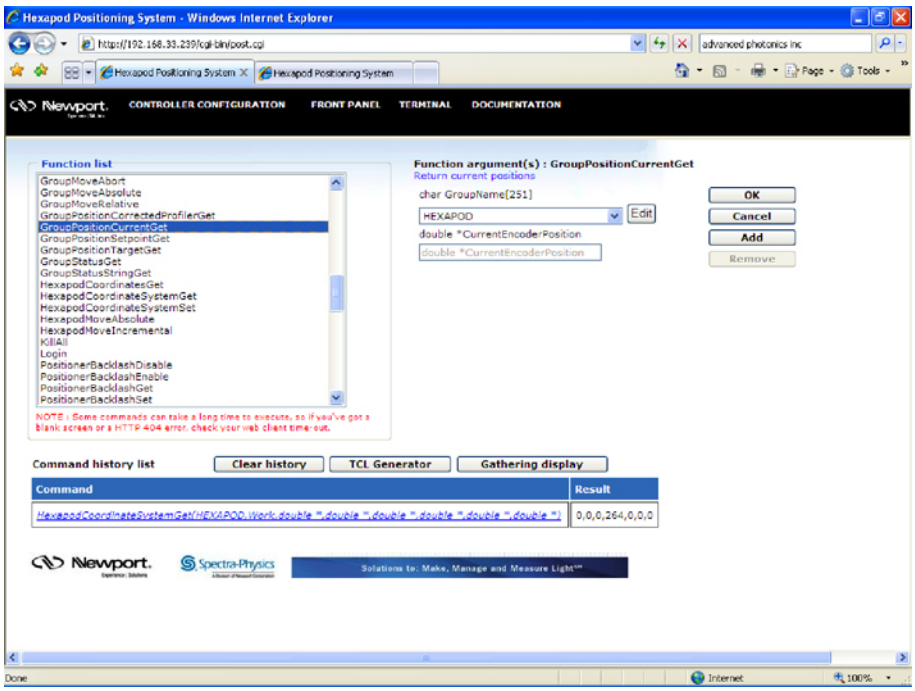
The terminal tool allows executing all functions of the HXP controller. It provides also a convenient method for generating executable TCL scripts. For more details about TCL scripts, see chapter 14.1.



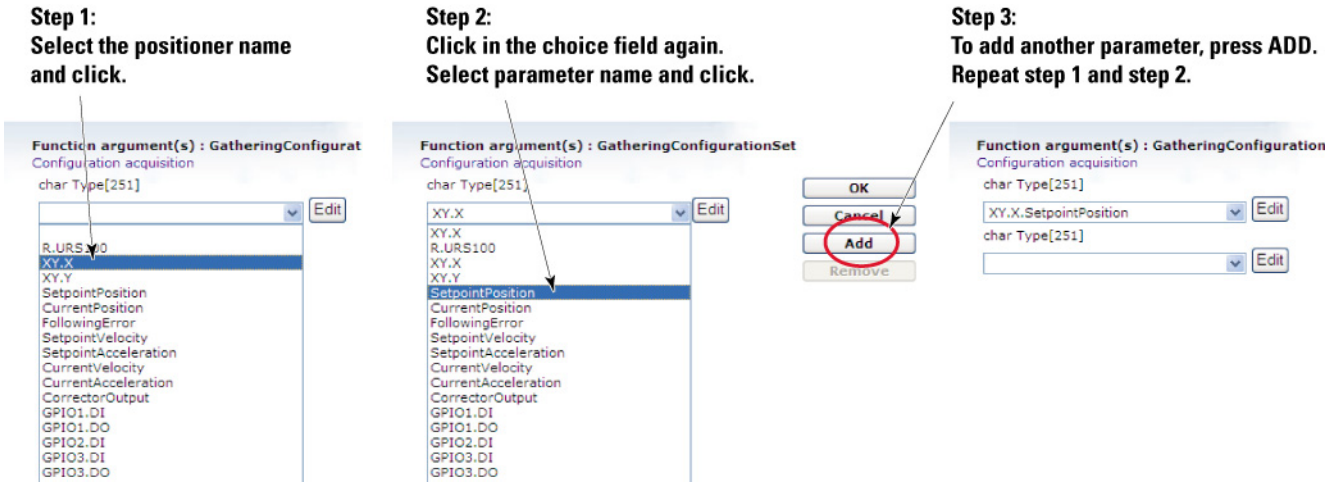
To execute a function from the Terminal, do the following:

1. Select a function and double-click to validate the selection.
2. Define the arguments for the function.

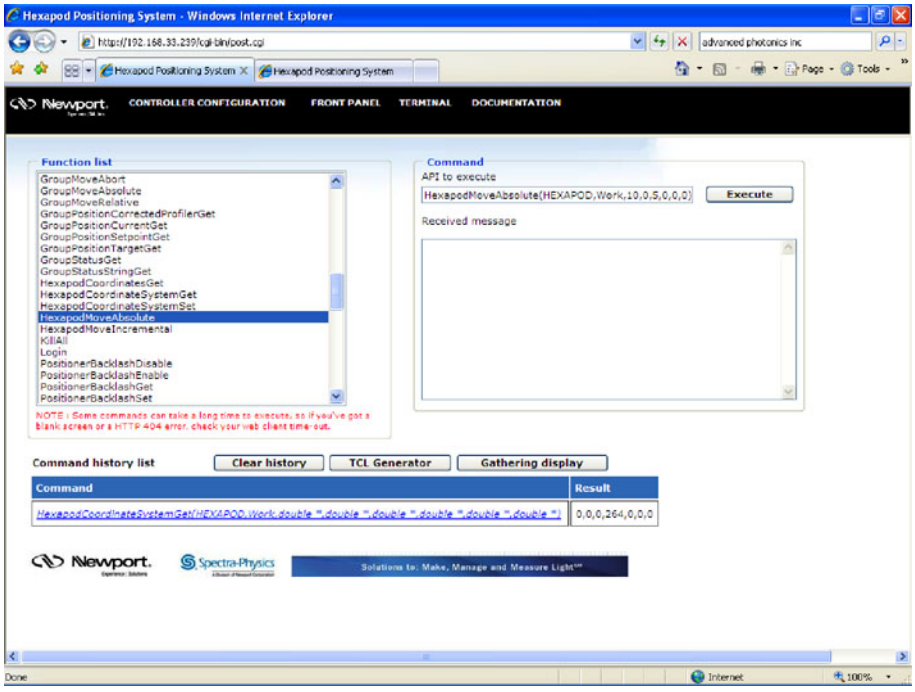
   For functions with dynamic arguments “ADD” and “REMOVE” buttons are available. Alternatively, you can use a “,” as separator between different arguments.



For some arguments like ExtendedEventName, ExtendedActionName or GatheringType, the argument name is not directly accessible. In these cases, define the first part of the argument name, then click in the choice field again and define the second part of the argument name. See example below for defining the GatheringType with the function GatheringConfigurationSet():

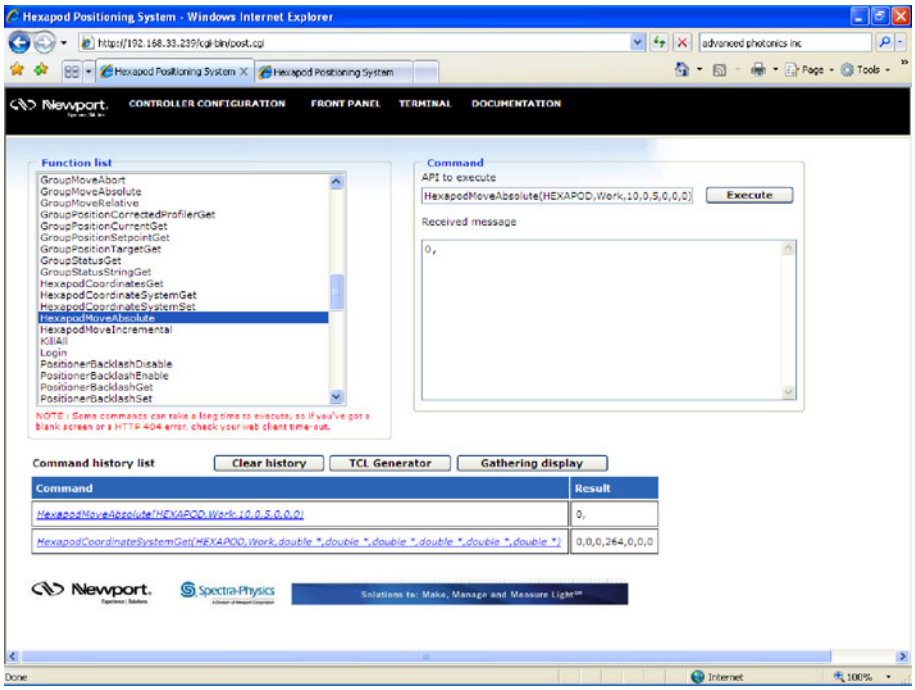


3. When all arguments are defined, click “OK”. Now you can review the final syntax of the function and can make final text changes, if needed. When done, click “Execute”.



4. When the function is executed, you’ll see the controller’s response in the Received message window. A returned 0 means that the function has been executed successfully. In all other cases, you’ll receive an error code. Use the function ErrorStringGet() to get more information about the error.

   For example, ErrorStringGet(-22) returns 0, Error -22 : Not allowed action.



NOTE

All functions are listed in alphabetic order. Only those functions listed are accessible in the current configuration of the HXP controller.

5.0 FTP (File Transfer Protocol) Connection

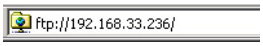
FTP is the protocol for exchanging files over the Internet. It works in the same way as HTTP for transferring web pages from a server to a user's browser and SMTP for transferring electronic mail across the Internet. FTP uses the Internet TCP/IP protocol to enable data transfer.

An FTP connection is needed to review the information inside the HXP controller, to download documentation, to transfer configuration files (to modify them directly), to transfer TCL scripts, etc…

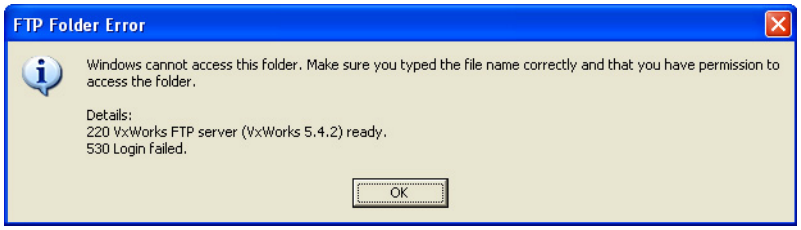
To connect to the FTP server:

- Start the HXP controller and wait until the boot sequence completes.
- Open an Internet browser window.
- Connect to the FTP server with the IP address of the controller:

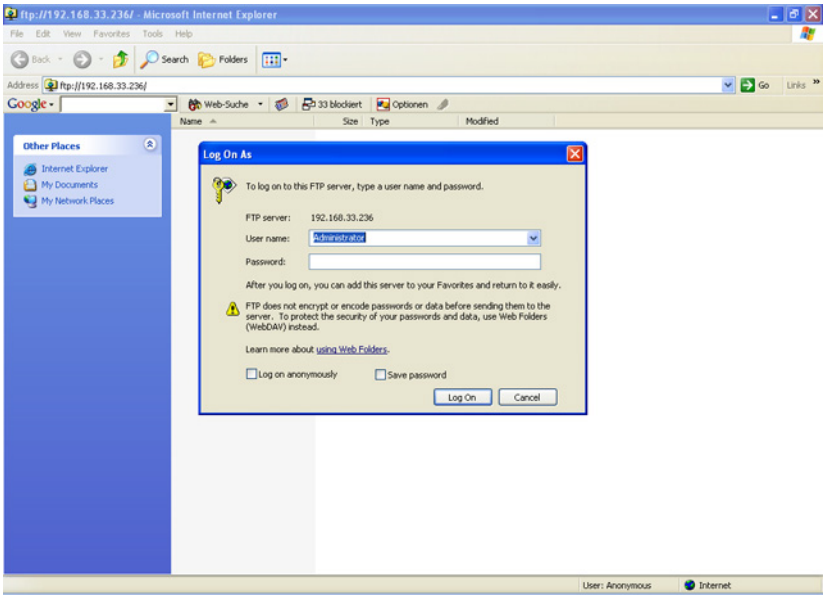
Example



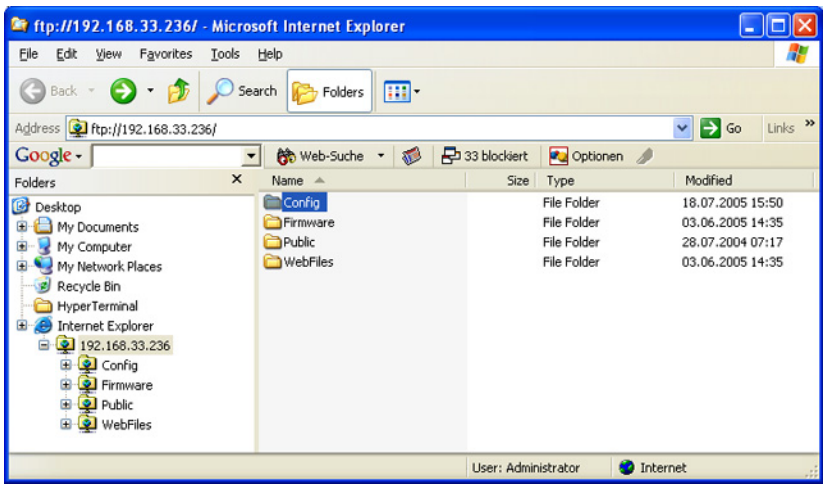
- It is normal that the following error message appears:



- Press OK.
- Select “File” from the top menu of the Internet browser, and then “Connect as…”. The following window appears:



Specify the HXP user name and HXP password. Press log on. The folders of the HXP controller are displayed (see below). Browse through the different folders and transfer data from or to your host PC the same way as with Windows Explorer.



6.0 Maintenance and Service

6.1 Enclosure Cleaning

WARNING

The HXP Controller/Driver should only be cleaned with a sufficient amount of soapy water solution. Do not use acetone or alcohol solutions, this will damage the finish of the enclosure.

6.2 Obtaining Service

The HXP Controller/Driver contains no user serviceable parts. To obtain information regarding factory service, contact Newport Corporation or your Newport representative and be ready with the following information:

- Instrument model number (on front panel).
- Instrument serial number (on rear panel) or original order number.
- Description of the problem.

If the instrument is to be returned to Newport Corporation, a Return Number will be issued, which should be referenced in the shipping documents.

Complete a copy of the Service Form as shown at the end of this User's Manual and include it with your shipment.

6.3 Troubleshooting

For troubleshooting, the user can query different error and status information from the controller. The HXP controller can provide the Positioner Error, the Positioner Hardware Status, the Positioner Driver Status, the Group Status, and also a general system error.

If there is an error during command execution, the controller will return an error code. The function ErrorStringGet() can be used to retrieve the description belonging to the error code.

The following functions are used to retrieve the Positioner Error and the Positioner Hardware Status:

- PositionerErrorGet(): Returns an error code.
- PositionerErrorStringGet(): Returns the description of the error code.
- PositionerHardwareStatusGet(): Returns the status code.
- PositionerHardwareStatusStringGet(): Returns the description belonging to the status code.

In a fault condition, it is also very important to know the current status of the group and the cause of the transition from the previous group status to the current group state. The following functions can be use to retrieve the Group Status:

- GroupStatusGet(): Returns the group status code.
- GroupStatusStringGet(): Returns the description belonging to the group status code.

NOTE

Refer to the Programmer's Manual for a complete list of status and error codes. Also refer to chapter 4.0 for troubleshooting the HXP controller with the help of the Newport web utilities.

6.4 Updating the Firmware Version of Your HXP Controller

Users can regularly update the controller with new firmware releases. To review the current available versions, refer to the FTP server:

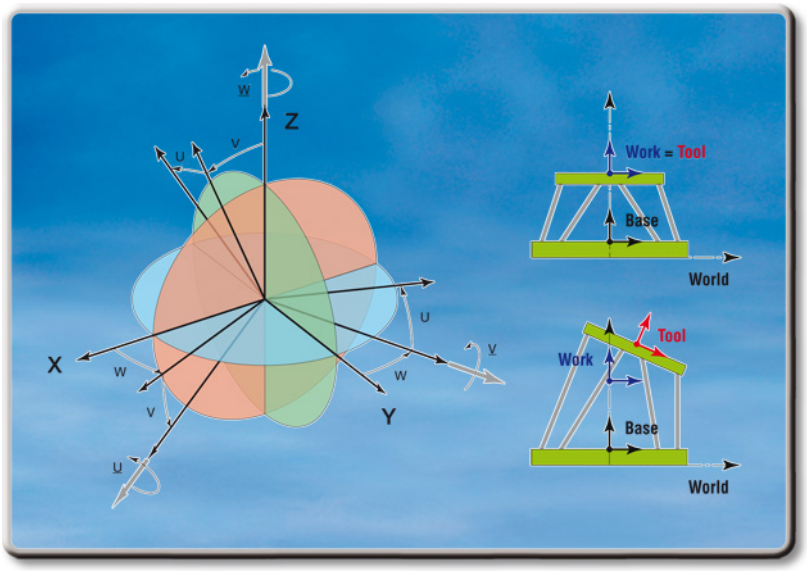
ftp://download.newport.com/MotionControl/Current/MotionControllers/HXP/

There are separate folders for the different versions, each folder contains the Firmware pack, the RC IHM and the Stage data base.

To install a new version, follow these instructions:

1. Double click on the HXP_Install.exe file (in FirmwarePack) and follow the instructions without rebooting the controller.
2. Double click on the StageDBInstall.exe file (in StageDataBase), follow the instructions and reboot the controller.
3. Uninstall both installers. To do so, click on Windows Start menu → Programs → Newport → HXP Tools → Firmware Pack FTP Updater → Uninstall Firmware Pack FTP Updater, then click on "Accept", and "OK".

A history file for the firmware and the stage database is added to the web documentation.



Motion Tutorial

This motion tutorial provides a detailed discussion of all features. It is written as a learning book with many application examples. The descriptions of the individual features are much more exhaustive than the short descriptions provided in the programmer's manual. It is highly recommended that all users read this motion tutorial before developing their applications.

The HXP Hexapod controller has been derived from Newport's Universal High-Performance XPS Motion Controller/Driver. HXP and XPS share the same hardware, but HXP features a special firmware for Hexapods. The fundamental firmware architecture of XPS and HXP are common, though, and HXP has inherited many "functions" from the XPS (functions are sometimes in the literature also referred to as software commands or API's). We want to mention this explicitly, because some of the function names of the HXP may sound odd or overcomplicated, but they have their origin in the firmware of the XPS, that has much broader usage.

7.0 HXP Architecture

7.1 Motion Groups

There are two hexapod controller versions: one with hardware for 6 axes (Hexapod Group only) and another with hardware for 8 axes (Hexapod Group plus 1 or 2 SingleAxis Groups). Within the HXP controller, each positioner or axis of motion must be assigned to a motion group. This "group" can either be a Hexapod group or a SingleAxis group depending on the hardware configuration and a system.ref file parameter enabled. A Hexapod group, for instance, is a motion group that features 6 positioners requiring at least a six axes controller. The 6 positioners are the 6 struts or actuators of the Hexapod. A Hexapod group and up to two Single Axis groups are the only motion groups that can be defined at present time for the HXP, but the XPS controller knows other motion groups such as a Spindle group, an XY group, an XYZ group or a MultipleAxes group. This is the reason for this broader group structure.

The functions of the HXP controller (again, functions are sometimes also referred to as software commands or API's in the literature) are grouped by categories. A category can be a positioner, a group, or a specific group like a Hexapod. There are also categories not directly related to a motion group such as functions for Analog and Digital IO's, for Data Gathering, for TCP/IP communication, and for Events and actions, among others.

The function **PositionerBacklashSet (PositionerName, Value)**, for instance, is a function for a "positioner". It requires a PositionerName and a value as input parameters and sets the value for the Backlash compensation of a positioner. A "positioner" can be one of the six struts of a Hexapod group.

The function **GroupHomeSearch (GroupName)** is a "group" function. It requires a "GroupName" as input parameter and launches a HomeSearch routine for the entire motion group. If "GroupName" refers to a Hexapod group, it launches a HomeSearch of the entire Hexapod.

The function **TCLScriptExecute (TCLFileName, TaskName, InputArguments)** is a "TCL function". It requires a TCLFileName and a TaskName as Input parameters, and InputArguments depending on the TCLFile. It executes of a TCL script.

The "GroupNames" and the "PositionerNames" are set in the system.ini configuration file of the HXP that is stored in the "..admin\config" folder of the HXP controller. Here is an example of a system.ini file with one Hexapod group and one linear stage:

```
[GENERAL]
BootScriptFileName =
BootScriptArguments =

[GROUPS]
HexapodInUse = HEXAPOD
SingleAxisInUse= Single1

;-------------------------------------------------------------------------

[HEXAPOD]
GeometryFileName = GeometrySTQ_RRPS.ini

PositionerInUse = 1,2,3,4,5,6
InitializationAndHomeSearchSequence = Together

WorkCoordinates = 0, 0, 196, 0, 0, 0
BaseCoordinates = 0, 0, 25, 0, 0, 0
ToolCoordinates = 0, 0, 25, 0, 0, 0

[HEXAPOD.1]
PlugNumber = 1
StageName = LTA1@LTA-HX@XPS-DRV01

[HEXAPOD.2]
PlugNumber = 2
StageName = LTA2@LTA-HX@XPS-DRV01

[HEXAPOD.3]
PlugNumber = 3
StageName = LTA3@LTA-HX@XPS-DRV01

[HEXAPOD.4]
PlugNumber = 4
StageName = LTA4@LTA-HX@XPS-DRV01

[HEXAPOD.5]
PlugNumber = 5
StageName = LTA5@LTA-HX@XPS-DRV01

[HEXAPOD.6]
PlugNumber = 6
StageName = LTA6@LTA-HX@XPS-DRV01

;-------------------------------------------------------------------------
```

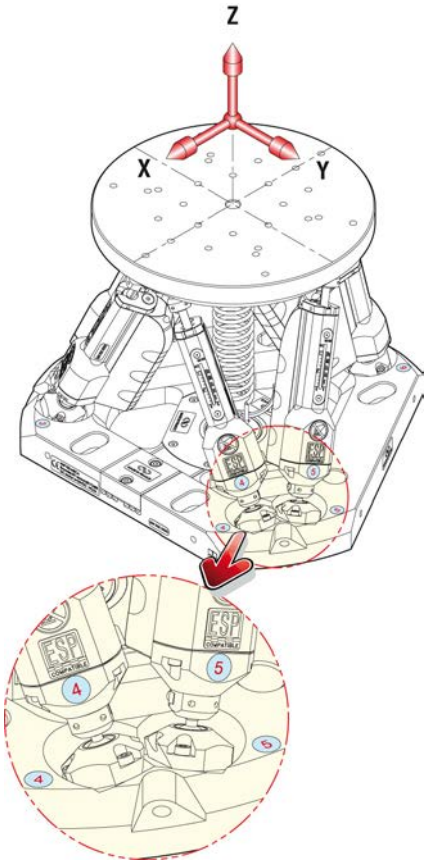
```
[SINGLE1]
PositionerInUse = POS

[SINGLE1.POS]
PlugNumber = 7
StageName = ILS200LM@XPS-DRV02
EncoderHardInterpolatorErrorCheck = Disabled
;--- Time flasher
TimeFlasherBaseFrequency = 40e6
;--- CIE08CompensatedPCO mode
CIE08CompensatedPCOMode = Disabled ; Enabled or Disabled
;--- PID Base filter
PIDBaseFilter = Disabled
```

In this example, the GroupName of the Hexapod group is HEXAPOD and the group name for the linear stage is SINGLE1. The 6 positioners of HEXAPOD are HEXAPOD.1, HEXAPOD.2, HEXAPOD.3, HEXAPOD.4, HEXAPOD.5, and HEXAPOD.6 and the positioner name for the linear stage is SINGLE1.Pos.

HXP manages all safeties and trajectories on a group level. For instance, the function **GroupHomeSearch (HEXAPOD)** homes the whole HEXAPOD group. In this case all six struts of the Hexapod are moved to their reference position in a well defined sequence that is also defined in the systerm.ini file (“Together” in the example above). In case of any error during this homing process, for instance if the home position of one strut is not found, the whole home process is stopped and the motion group goes to an error state (see next section).

7.2 Hexapod Group

7.2.1 Hexapod Coordinate Systems



A Hexapod is a motion device with 6 degrees of freedom. The Hexapod platform can translate in three perpendicular axes, referred to as X, Y and Z, and rotate around these axes, referred to as U, V, and W. U, also referred to as roll, is a rotation around X. V, also referred to as pitch, is a rotation around Y. And W, also referred to as yaw, is a rotation around Z. Any position of the Hexapod platform is described by these six coordinates: (X Y Z U V W).

For translations, HXP uses Cartesian coordinates designated as X, Y, and Z. X usually refers to the main propagation axis, in optics for instance to the direction of the optical rays. Z refers to the vertical axis. X, Y and Z form a right-handed coordinate system.

For rotations, HXP uses Bryant angles, sometimes in the literature also referred to as Tait-Bryan angles, Cardan angles, or Euler angles in definition ZYX. This definition is frequently used in Hexapod motion, robotics and aviation.

So how is the position (X Y Z U V W) of a HXP Hexapod group defined? Consider a *xyz* coordinate system, where the moving carriage of the Hexapod platform is attached to, meaning, the carriage of the Hexapod moves with the coordinate system. This is called the **Tool** coordinate system.. Consider a second coordinate system XYZ that is fixed, means it does not move with the carriage. This is called the **Work** coordinate system. The six coordinates (X Y Z U V W) describe the position of the Tool coordinate system **in** the Work coordinate system. At position (0 0 0 0 0 0) the two coordinate systems, Tool and Work, are coincident. The position (X Y Z U V W) is reached as follow:

1. Starting from position (0 0 0 0 0 0), move the Tool coordinate system to the position XYZ in the Work coordinate system
2. Make a clockwise rotation about the *z* axis of the Tool coordinates system by the yaw angle (W).
3. Make a clockwise rotation about the new (once rotated) *y* axis of the Tool coordinate system by the pitch angle (V)

4. Make a clockwise rotation about the new (twice rotated) *x* axis of the Tool coordinate system by the roll angle (U)

Figure 13 illustrates the angular motion. Figure 14 is a simpler graphic with the example of an airplane as object.

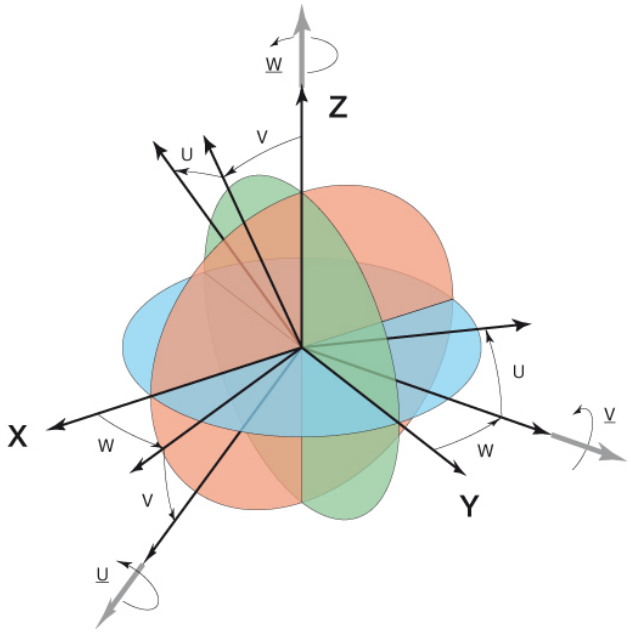


Figure 12: Bryant angle defintion.

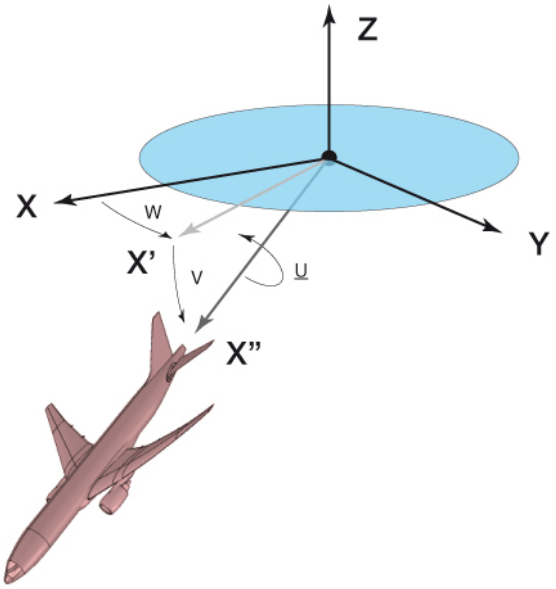


Figure 13: Bryant angle definition (simple representation).

So, the important thing to remember is:

> **The position of the Hexapod moving carriage is always understood as the position of the Tool coordinate system in the Work coordinate system**.

The tool coordinate system is attached to the carriage and moves with the carriage and vice versa. The Work coordinate system is stationary and is defined in the “World coordinate system”. By changing the length of the six Hexapod struts, the position of the Tool coordinate system in the work coordinate system is changed.

However, this information is not sufficient to fully define the position of the Hexapod carriage in the world coordinate system. This is because it is also possible to move the

Hexapod at its base plate to a new location. In this case, the position of the Hexapod carriage changes as well, yet the length of the struts haven't changed. Hence we need another, a third coordinate system that defines the position of the Hexapod base in the World coordinate system. This is called the **Base** coordinate system.

Base coordinate System

The Hexapod is physically located in the Base coordinate system so that all lower joints of the Hexapod struts are in the XY plane of the Base coordinate system (Z=0), the center line of the Hexapod matches with the Z-axis, and the motor cables face in the X-direction (see also technical drawing provided with the Hexapod mechanics).

The position of Base coordinate system in the world coordinate system is defined in the system.ini configuration file. The system.ini file is located in the "..admin\config" folder of the HXP controller. On page 38 you can see an example of such a configuration file. The position of the Base coordinate system can only be changed by modifying the values in the configuration file and not by function call. In the default configuration of the HXP controller, the Hexapod is located as follows in the world coordinate system:

- The Hexapod base plate is orthogonal to the Z-axis of the world coordinate system
- The center of the lower surface of the Hexapod base plate has the world coordinates X=0, Y=0, Z=0
- The motors cables point in the X-direction of the world coordinate system.

That means, the only coordinate of the Base coordinate system in the world coordinate system that is different from zero is the Z-coordinate, and this Z-coordinate is equal to the distance between the plane of the lower joints and the lower surface of the Hexapod.

For those to whom this sounds too complicated, don't change with the Base coordinate system in the system.ini file, but keep in mind the following:

> **In the default configuration of the HXP, the center of the world coordinate system is at the center of the lower surface of the Hexapod base plate, Z is orthogonal to the base plate, and X is in the direction of the motor cables.**

So why is it necessary to change the position of the Base system at all? The primary need is if the position of the Base has changed, for instance after service or if the Hexapod has been exchanged with another one with a slightly different geometry. In this case it is possible to change the position of the Base coordinate system in the system.ini file without having to make other changes. Any application can run exactly the same with the exact same physical positions being reached, yet the length of the Hexapod struts could be different to reach the same position in space. All this will be managed by the HXP.

Again, for the average user, this may not be important and the only thing to bear in mind is the following:

> **In the default configuration of the HXP, the center of the world coordinate system is at the center of the lower surface of the Hexapod base plate, Z is orthogonal to the base plate, and X is in the direction of the motor cables.**

Tool coordinate system

The Tool coordinate system is attached to the carriage and moves with the carriage, and vice versa. So we need to define the position of the Tool coordinate system relative to the physical properties of the Hexapod carriage. This is done by another coordinate system, that we call "Carriage". This Carriage system is physically located so that all upper joints of the Hexapod struts are in the XY plane of the Carriage coordinate system (Z=0), the center line of the Hexapod carriage matches with the Z-axis, and the directions of X and Y are the same as the ones of the world coordinate system when the Hexapod is at its reference home position.

The position of Tool coordinate system is set in the system.ini configuration file, but its position can also be changed by function call, see 9.4.6 for details. In the default configuration of the HXP controller, the Tool coordinate system is set as follow:

- The center of the Tool coordinate system matches with the center of the upper surface of the Hexapod carriage. In other words, the center of the upper surface of the Hexapod carriage has the Tool coordinates X=0, Y=0, and Z=0.
- The Z-axis of the Tool coordinate system is orthogonal to the carriage
- The orientation of the X- and Y-axis of the Tool system are the same as the ones of the Base coordinate system (and hence also of the World coordinate system), when the Hexapod is at its reference Home position.

That means the only coordinate of the Tool coordinate system (in the carriage coordinate system) that is different from zero is the Z-coordinate and this Z-coordinate is equal to the distance between the plane of the upper joints and the upper surface of the Hexapod carriage.

For those to whom this sounds too complicated, just bear in mind the following:

In the default configuration of the HXP, the center of the Tool coordinate system is at the center of the upper surface of the Hexapod carriage, Z is orthogonal to the carriage, and X is in the direction of the motor cables, when the Hexapod is at its reference Home position.

Work coordinate system

The Work coordinate system is defined in the World coordinate system (remember, by default, the center of the World system is at the center of the lower surface of the Hexapod base, Z is orthogonal to the base, an X is in direction of the motor cables).

In the default configuration of the HXP, the Work coordinate system is located as such that the Tool coordinate system is coincident with the Work coordinate system when the Hexapod is at its reference Home position.

In other words, the Hexapod has the position (0 0 0 0 0 0) at the reference Home position (remember: position of a Hexapod is always position of the Tool system in the Work system).

The position of the Work coordinate system (in the World system) is set in the system.ini configuration file, but its position can also be changed by function call, see chapter 9.4.6 for details.

Figure 15 provides simple 2D sketches of the locations of the different coordinate systems in the default configuration of the HXP.

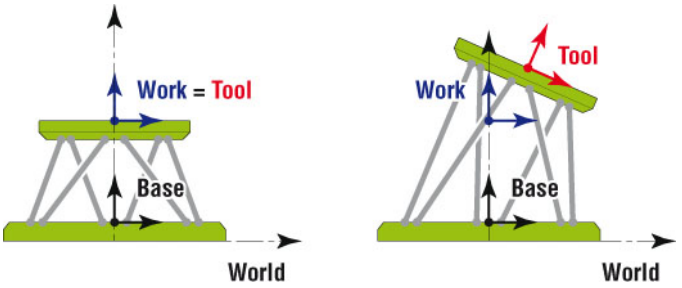


Figure 14: HXP default coordinate systems at home position (left) and any other position (right). Note that the tool coordinate system always moves with the carriage.

So before moving forward we want to summarize the most important things:

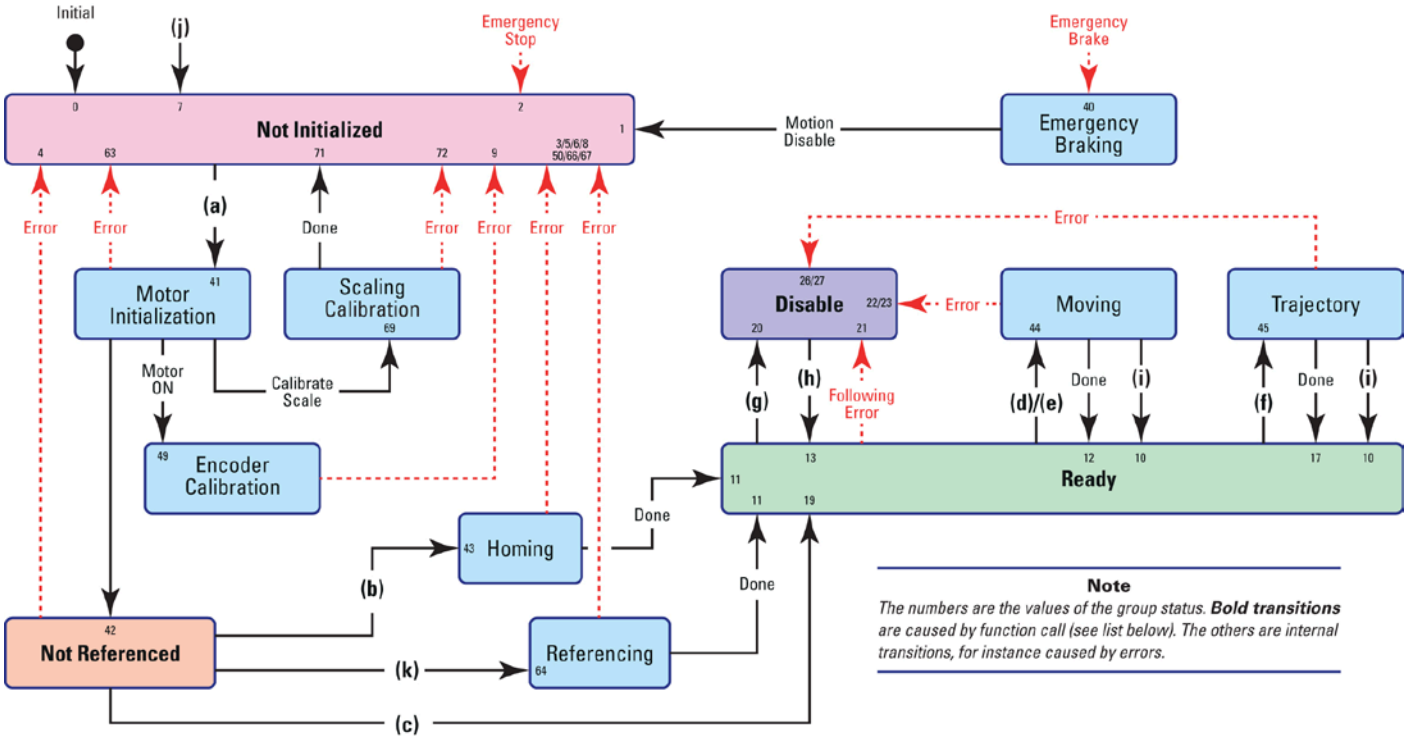
1. **The position of the Hexapod (moving carriage) is understood as the position of the Tool coordinate system in the Work coordinate system.**
2. **The tool coordinate system is attached to the carriage and moves with the carriage, and vice versa. In the default configuration of the HXP, the center of the Tool system is at the center of the upper surface of the Hexapod carriage, Z is orthogonal to the carriage, and X is in the direction of the motor cables, when the Hexapod is at its reference Home position.**
3. **The work coordinate system is stationary. In the default configuration of the HXP, the Work coordinate system is coincident with the Tool coordinate**

system when the Hexapod is at its reference Home position. This means, the Hexapod has the position (0 0 0 0 0 0) after Homing.

4. **All positions are defined in Cartesian coordinates with Bryant angles. A position (X Y Z U V W) is reached as follow:**
   a) **Starting from position (0 0 0 0 0 0), move the Tool coordinate system to the position XYZ in the Work coordinate system.**
   b) **Make a clockwise rotation about the Z-axis of the Tool coordinates system by the yaw angle (W).**
   c) **Make a clockwise rotation about the new (once rotated) Y-axis of the Tool coordinate system by the pitch angle (V).**
   d) **Make a clockwise rotation about the new (twice rotated) X-axis of the Tool coordinate system by the roll angle (U).**

7.2.2 Hexapod State Diagram

The HXP uses group states to describe the "status" of each motion group. A group status is an integer value that is representative for the current status of a motion group and the reason for the last transition. All possible states and possible transitions of a motion group are summarized in the state diagram.



Status Changing Functions

(a) GroupInitialize	**(f)** HexapodMoveIncrementalControl
(b) GroupHomeSearch	**(g)** GroupMotionDisable
(c) GroupReadyAtPosition	**(h)** GroupMotionEnable
(d) GroupMoveAbsolute or GroupMoveRelative	**(i)** GroupMoveAbort
(e) HexapodMoveAbsolute or HexapodMoveRelative	**(j)** GroupKill or KillAll
	(k) GroupReferencingStart

Figure 15: State diagram of the Hexapod group.

After booting the HXP controller, all groups are in state 0, "Not Initialized State". The only possible function is a **GroupInitialize (GroupName)** that, if successful, transfers the group to state 42, "Not Referenced State". In the state 42 the motors are powered.

The next only possible function is a **GroupHomeSearch or referencing (MUST COME BACK TO THIS POINT AND ELABORATE) (GroupName)** that launches a Home search process for the motion group. After successful completion the motion group is in the state 11, “Ready State from Homing”. In case of an error during the execution of the GroupHomeSearch(), the group could be in the state 3, “Not initialized state due to a following error during homing” or state 6, “Not initialized state due to an end of run after homing”.

The state diagram visualizes also all state changing functions. For instance it is not possible to launch a GroupHomeSearch() when a group is in state 12, “Ready state from motion”. To do so, it is required to send a GroupKill(), followed by a GroupInitialize().

The group states are very helpful for error diagnostics. In case of any unexpected behavior, query the group status using the function **GroupStatusGet (GroupName)**. This function returns the current state number. The function **GroupStatusStringGet (StateNumber)** returns the description corresponding to the state number. A complete list of all group states can be found in chapter 2.13 of the HXP programmer’s manual.

In the consequences to an error, the HXP distinguishes between major faults and minor faults:

Error type	Fault	Consequence
General inhibition Motor fault Encoder fault	Major	Emergency stop
End of travel	Major	Emergency brake
Following error	Minor	Motion disable

- After an emergency brake or an emergency stop, both considered major faults, the group is in the “not initialized” state. The system has to be initialized and homed again.
- After a following error, as it is considered a minor fault, the group is in the “Disable” state. The function **GroupMotionEnable (GroupName)** gets the group back to the “ready” state.

7.3 SingleAxis Group

7.3.1 SingleAxis Enable Switch

The HXP controller requires a parameter "SingleAxisGroupOption" in the system.ref file to be switched enabled or disabled in order to implement the additional two SingleAxis groups.

- If "Disabled" then only one hexapod group can be configured (6 axes).
- If "Enabled" then it is possible to configure up to two additional SingleAxis groups (only possible with a controller equipped for 8 axes).

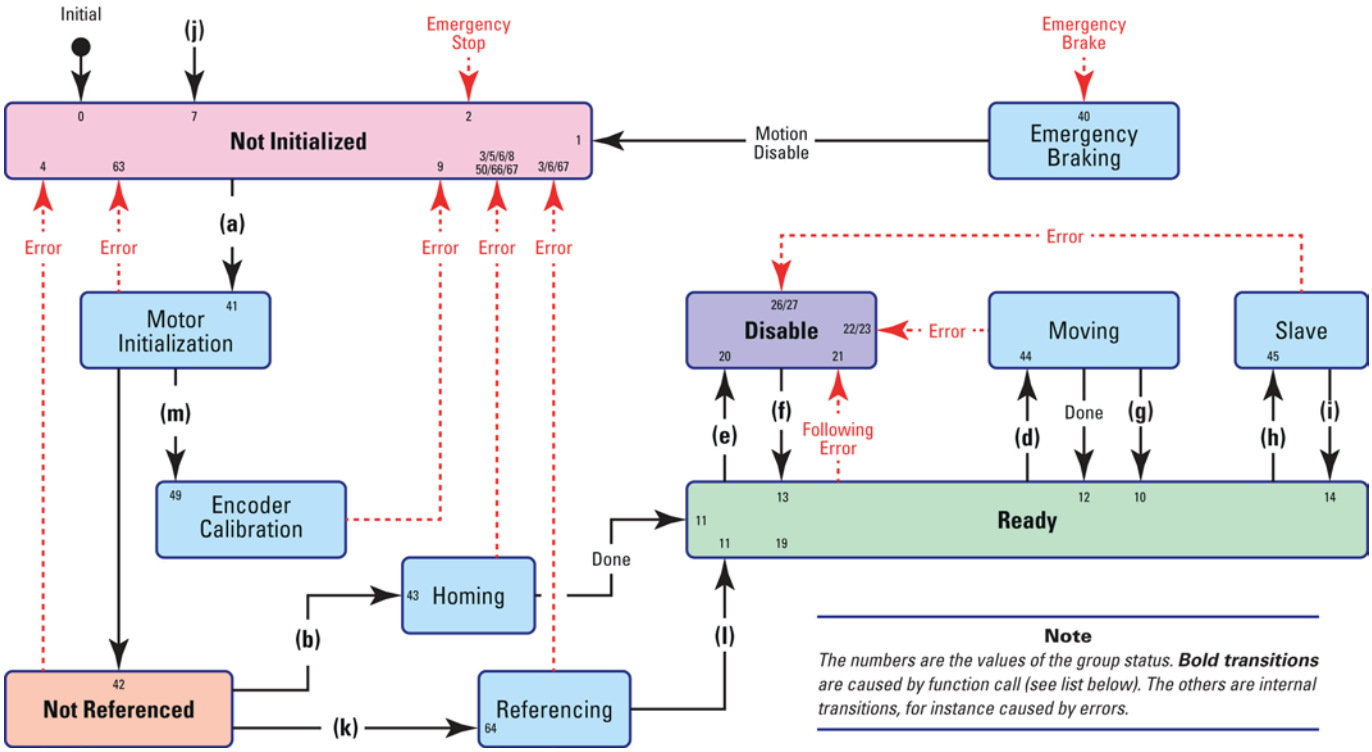
Sample "system.ref " file

```
[GENERAL]
FirmwareName = MainController
ExternalModuleNames =
CorrectorISRPeriod = 175e-6 ; Servo rate in seconds
IRQDelay = 6e-6 ; seconds
DACUpdateDelay = 154e-6 ; seconds
ProfileGeneratorISRRatio = 14 ;
ServitudesISRRatio = 10
GatheringBufferSize = 1000000
DelayBeforeStartup = 0 ; seconds
SingleAxisGroupOption = Enabled ; Enabled or Disabled - Enabled is only allowed on 8 axes
                                  system (HXP + 2)
```

The system.ref file is accessible through the FTP protocol connection described in chapter 5.0 FTP (File Transfer Protocol) Connection.

7.3.2 SingleAxis State Diagram

The HXP uses group states to describe the "status" of each motion group. A group status is an integer value that is representative for the current status of a motion group and the reason for the last transition. All possible states and possible transitions of a motion group are summarized in the state diagram.



Status Changing Functions

(a) GroupInitialize	**(g)** GroupMoveAbort
(b) GroupHomeSearch	**(h)** SpinSlaveModeEnable
(c) GroupReadyAtPosition	**(i)** SpinSlaveModeDisable
(d) GroupMoveAbsolute or GroupMoveRelative	**(j)** GroupKill or KillAll
(e) GroupMotionDisable	**(k)** GroupReferencingStart
(f) GroupMotionEnable	**(l)** GroupReferencingStop
	(m) GroupInitializationWithEncoderCalibration

State diagram of the SingleAxis group.

After booting the HXP controller, all groups are in state 0, "Not Initialized State". The only possible function is a **GroupInitialize (GroupName)** that, if successful, transfers the group to state 42, "Not Referenced State". In the state 42 the motors are powered. The next only possible function is a **GroupHomeSearch or referencing (MUST COME BACK TO THIS POINT AND ELABORATE) (GroupName)** that launches a Home search process for the motion group. After successful completion the motion group is in the state 11, "Ready State from Homing". In case of an error during the execution of the GroupHomeSearch(), the group could be in the state 3, "Not initialized state due to a following error during homing" or state 6, "Not initialized state due to an end of run after homing".

The state diagram visualizes also all state changing functions. For instance it is not possible to launch a GroupHomeSearch() when a group is in state 12, "Ready state from motion". To do so, it is required to send a GroupKill(), followed by a GroupInitialize().

The group states are very helpful for error diagnostics. In case of any unexpected behavior, query the group status using the function **GroupStatusGet (GroupName)**. This function returns the current state number. The function **GroupStatusStringGet**

(StateNumber) returns the description corresponding to the state number. A complete list of all group states can be found in chapter 2.13 of the HXP programmer's manual.

In the consequences to an error, the HXP distinguishes between major faults and minor faults:

Error type	Fault	Consequence
General inhibition Motor fault Encoder fault	Major	Emergency stop
End of travel	Major	Emergency brake
Following error	Minor	Motion disable

- After an emergency brake or an emergency stop, both considered major faults, the group is in the "not initialized" state. The system has to be initialized and homed again.
- After a following error, as it is considered a minor fault, the group is in the "Disable" state. The function **GroupMotionEnable (GroupName)** gets the group back to the "ready" state.

7.4 Function Error Codes

Every function sent to the HXP returns an error code upon completion. In case of a successful execution, the return is "0", "Successful command execution". In case of an error or an unsuccessful execution, the return is a negative integer value. The error code "-4", for instance, "unknown command", typically refers to a typographical error in the function name. The error code "-17", "Parameter out of range", can refer to a Move Command to a position that is outside of the allowed motion range. The function **ErrorStringGet (ErrorCode)** returns the description corresponding to the error code. A complete list of all function error codes can be found in chapter 2.14 of the HXP programmer's manual.

Please note, that HXP uses TCP/IP blocking sockets, which means that a socket is "blocked" while a function is executed. It is freed with the completion of the function by sending of the error code. As a consequence, a new function can be only executed on the same socket when the previous function is completed. In order to run several processes in parallel, different sockets and threads as needed. Please refer to section 16.4, "Running Processes in parallel" for further information on this subject.

8.0 Motion

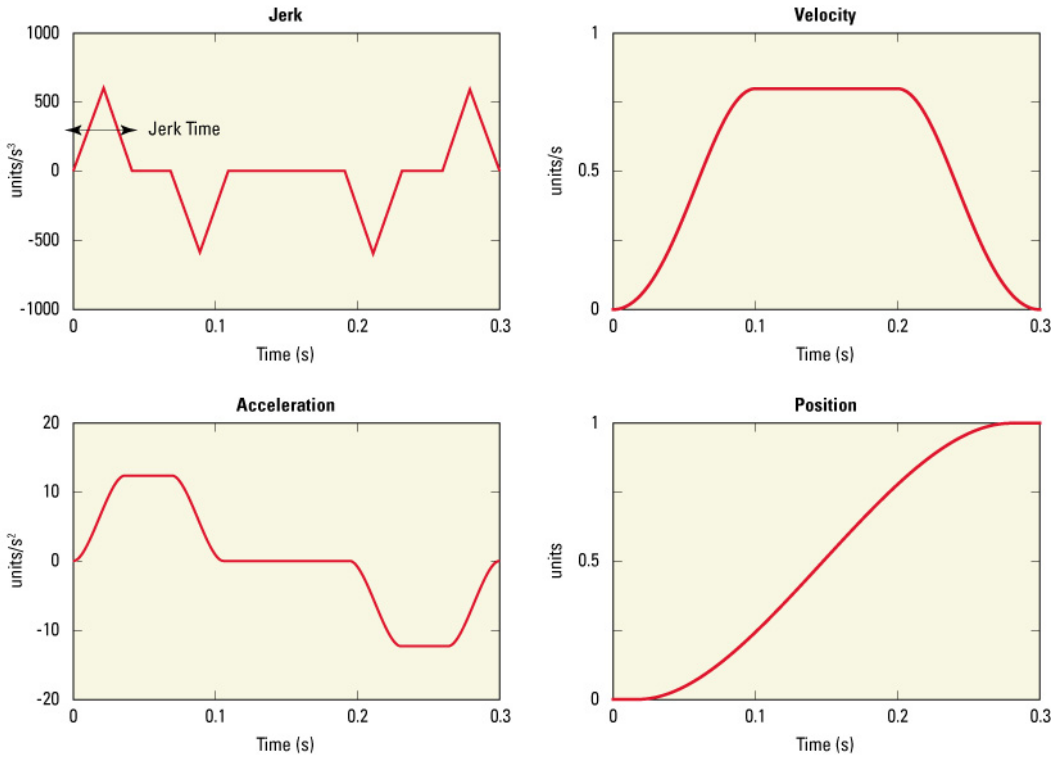
8.1 Measurement Units

The HXP controller is preconfigured so that all length units are in millimeters and all angular units are in degrees. This is true for the positions of the Hexapod group, the positions of the coordinate systems and all motion positioner related length units (such as a positioner is one of the struts of a Hexapod). This means all specifications like travels, speeds, accelerations, etc. have units of mm, mm/s, mm/s², etc.

8.2 Motion Profiles

Motion commands refer to strings sent to a motion controller that will initiate a motion. On execution of a motion command, the positioner moves from the current position to the desired destination. The exact trajectory for the motion is calculated by a motion profiler. So the motion profiler defines where each of the positioners should be at each point in time. There are details worth mentioning about the motion profiler in the HXP controller:

In a classical trapezoidal motion profiler (trapezoidal velocity profile), the acceleration is an abrupt change. This sudden change in acceleration can cause mechanical resonance in a dynamic system. In order to eliminate the high frequency portion of the excitation spectrum generated by a conventional trapezoidal velocity motion profile, the HXP controller uses a sophisticated SGamma motion profile. Figure 17 shows the acceleration, velocity and position plot for the SGamma profile.



Displacement: 150 e^{-3} units
Maximum velocity: 0.8 units/s
Maximum acceleration: 12 units/s²
Minimum jerk time: 0.004 s
Maximum jerk time: 0.04 s

Notice: The minimum displacement lasts at least 4 times the minimum jerk time.

Figure 16: SGamma Motion Profile.

The SGamma motion profile provides better control of dynamic systems. It allows for perfect control of the excitation spectrum that a move generates. In a multi-axes system this profile gives better control of each axis independently, but also allows control of the cross-coupling that are induced by the combined motion of the axes. As shown in figure 17, the acceleration plot is parabolic. The parabola is controlled by the jerk time (jerk being the derivative of the acceleration). This parabolic characteristic of the acceleration results in a much smoother motion. The jerk time defines the time needed to reach the necessary acceleration. One feature of the HXP controller is that it automatically adapts the jerk time to the step width by defining a minimum and a maximum jerk time. This auto-adaptation of the jerk time allows a perfect adjustment of the system's behavior with different motion step sizes.

NOTE

Because of jerk-controlled acceleration, any move has a duration of at least four times the jerk time.

For the HXP controller, the following parameters need to be configured for the SGamma profile:

- MaximumVelocity (units/s)
- MaximumAcceleration (units/s^2)
- EmergencyDecelerationMultiplier (Applies to Emergency Stop)
- MinimumJerkTime (s)
- MaximumJerkTime (s)

The above parameters are set in the stages.ini file for a positioner. When using the XPS controller with Newport stages, these parameters are automatically set during the configuration of the system.

The velocity, acceleration and jerk time parameters is modified by the function **PositionerSGammaParametersSet().**

Example

PositionerSGammaParametersSet (MyGroup.MyStage, 10, 80, 0.02, 0.02)

This function sets the positioner "MyStage" velocity to 10 units/s, acceleration to 80 units/s^2 and minimum and maximum jerk time to 0.02 seconds. The set velocity and acceleration must be less than the maximum values set in the stages.ini file. These parameters are not saved if the controller is shut down. After a re-boot of the controller, the parameters will retain the values set in the stages.ini file.

In actual use, the HXP places a priority on the displacement position value over the velocity value. To reach the exact position, the speed of the positioner may vary slightly from the value set in the stages.ini file or by the **PositionerSGammaParametersSet** function. So the drawback of the SGamma profile is that the velocity used during the move can be a little bit different from the velocity defined in the parameters. For example, the exact velocity will change when the move distance is changed, move 100mm, then 100.001 mm then 100.011 mm. There will be some changes to the commanded velocity. This change can be ignored for many applications except where an accurate time synchronization during the motion is required.

The function, **PositionerSGammaExactVelocityAdjustedDisplacementGet(),** can be used as described below to achieve the exact desired speed in applications that require an accurate value of the velocity during a move. In this case, the velocity value is adhered to, but the target position may be slightly different from the one required. In other words, according to the application requirements, the user can choose between very accurate positions or very accurate velocities.

Example

PositionerSGammaExactVelocityAdjustedDisplacementGet (MyGroup.MyStage, 50.55, ExactDisplacement)

This function returns the exact displacement for that move with the exact constant velocity set shown in the example above (10 mm/s). The result is stored in the variable ExactDisplacement, for instance 50.552.

GroupMoveAbsolute (MyGroup.MyStage, 50.552)

In the above example, for a position of 50.55 mm, the command returns a value of 50.552. This means that in order for the positioner "MyStage" to achieve the desired velocity in the most accurate way, the commanded position should be 50.552 mm instead of 50.55 mm.

The HXP can report two different positions. The first one is the SetpointPosition or theoretical position. This is the position where the stage should be according to the profile generator.

The second position is the CurrentPosition. This is the actual position as reported by the positioner's encoder after taking into account all compensation. The relationship between the SetpointPosition and the CurrentPosition is as follows:

Following error = SetpointPosition - CurrentPosition

The functions to query the SetpointPosition and the CurrentPosition values are:

GroupPositionCurrentGet() and GroupPositionSetpointGet()

8.3 Home Search

Upon power-up of the HXP controller, any group must be first initialized and homed.

The function **GroupInitialize (GroupName)** initializes the group. When initialized, the motors are powered.

The default GroupName for a Hexapod group is HEXAPOD (case sensitive). The name is defined in the system.ini configuration file of the HXP controller, see chapter 7.1 for details.

The function **GroupHomeSearch (GroupName)** launches a Home search process on the group. Homing refers to a predefined motion process that moves a stage to a unique reference position and defines this as Home. During the Hexapod home search process, all positioners move to a unique reference position, which, depending on the hardware, in most cases is close to the negative travel limit of the Hexapod struts.

The type of Home Search process used for each positioner is defined in the stages.ini configuration file of the HXP, that is stored in the "..admin\config" folder of the HXP controller. For more details about HomeSearch entries in the stages.ini in general, refer to the XPS controller manual. Typically, these settings are not to be modified by users.

In the default configuration of the HXP controller, the position of the Hexapod after homing is (0 0 0 0 0 0). This means the Tool coordinate system is coincident with the Work coordinate system.

NOTE

When the home position of a Hexapod is close to the minimum travel limit of the Hexapod struts, the only possible motion after homing is in the positive Z-direction.

A Hexapod has the maximum available travel range in X, Y, Z, U, V, and W when the Hexapod struts are approximately half way extended. This is the case when the Hexapod is approximately in the middle of its available Z travel range. So depending on the application, it may be recommended to move to this position after homing and to use this position as the "optimum working position". See also further note in chapter 9.7 about changing the positions of the Tool and Work coordinate systems.

Example

After homing all Hexapod struts are close to their minimum travel position. The position at home is (0 0 0 0 0 0) (default HXP configuration). The Hexapod has a Z travel range of 28 mm (+/-14 mm). The function **HexapodMoveAbsolute (HEXAPOD, Work, 0, 0, 14, 0, 0, 0)** moves the Hexapod to the absolute position (0 0 14 0 0 0). From this position, the Hexapod has a maximum possible travel range in X, Y, Z, U, V, and W.

8.4 Hexapod Motion

A move is a point-to-point motion. On execution of a move command, the motion device moves from a current position to a desired destination (absolute move) or by a defined increment (relative move). During motion, the controller is monitoring the feedback of the positioner and is updating the output based upon the following error. The HXP controller's position servo loop is being updated at up to 10 kHz and the profile generator at up to 2.5 kHz, providing highly accurate closed loop positioning. Between the profiler and the corrector (servo loop), there is a time-based linear interpolation to accommodate the different frequencies.

There are two types of moves that can be commanded: an absolute move and a relative move. For an absolute move, the positioner will move relative to the HomePreset position as defined in the stages.ini file. In most cases the HomePreset is 0, which makes the home position equal to the zero position of the positioner. For a relative move, the positioner will move relative to the current TargetPosition. In relative moves, it is possible to make successive moves that are not equal to a multiple of an encoder step without accumulating errors.

The functions for absolute and relative motions for the Single Axis Group(s) are **GroupMoveAbsolute()** and **GroupMoveRelative()** respectively.

Example

A motion system consisting of one SingleAxis group called ScanTable and one SingleAxis group called FocusStage.

…

GroupHomeSearch (ScanTable)

GroupHomeSearch (FocusStage)

After homing is completed…

GroupPositionCurrentGet (ScanTable, Pos1)

… will return 0 to Pos1, assuming PresetHome = 0.

GroupPositionCurrentGet (FocusStage, Pos2)

Will return 0 to Pos2, assuming HomePreset = 0.

GroupMoveAbsolute (ScanTable, 100)

GroupPositionCurrentGet (ScanTable, Pos1)

… will return 100 to Pos1.

GroupMoveRelative (FocusStage, 1)

GroupMoveRelative (FocusStage, 1)

The second move can be only executed after the first move is completed. After all moves are completed…

GroupPositionCurrentGet (FocusStage, Pos2)

… will return 2 to Pos2.

The velocity, acceleration and jerk time parameters of a move are defined by the function PositionerSGammaParametersSet() (see also section 9.2). When the controller

receives new values for these parameters during the execution of a move, it will not take these new values into account on the current move, but only on the following moves.

A move can be stopped at any time with the function **GroupMoveAbort()** that accepts GroupNames and PositionerNames. It is important to note, however, that the function **GroupMoveAbort(PositionerNames)** is accepted when the motion was commanded to the positioner, and not to the group. In the previous example, the function **GroupMoveAbort(ScanTable.ScanAxis)** is rejected for a motion that has been launched with **GroupMoveRelative(ScanTable, 100, 50)**. To stop this motion, send the function **GroupMoveAbort(ScanTable)**.

8.4.1 Hexapod Referencing State: GroupReadyAtPosition

A standard Hexapod home search process might not be desirable in the following situations.

- There is a risk of collision by moving a Hexapod to the default home position after re-initializing the motors.
- The system is in 'NOT INITIALIZED' state due to a non-permanent fault condition but encoders are still working.
- The position of a Hexapod must be maintained after the scheduled power shutdown.

The HXP controller's **GroupReadyAtPosition (GroupName)** is an alternative to the predefined Hexapod home search processes. It allows the Hexapod to be at READY state at the user-defined coordinates without performing a home search. The group must be initialized and must be in "NOT REFERENCED" state before executing the **GroupReadyAtPosition** function.

Example

Obtain and record the current position for each Hexapod strut.

GroupPositionCurrentGet(HEXAPOD.1, double *)

GroupPositionCurrentGet(HEXAPOD.2, double *)

GroupPositionCurrentGet(HEXAPOD.3, double *)

GroupPositionCurrentGet(HEXAPOD.4, double *)

GroupPositionCurrentGet(HEXAPOD.5, double *)

GroupPositionCurrentGet(HEXAPOD.6, double *)

Example of actuator (strut) positions

Note

User to document these actuator positions for later reference. Also document the last known hexapod absolute position (X,Y,Z,Tx,Ty, Tz) prior to shutdown.

After a power shutdown or when motors are turned off; the group is in NOT INITIALIZED state. When initialized, the hexapod should not move.

GroupInitialize(HEXAPOD)

Use the GroupReadyAtPosition to avoid the home search routine and to move the Hexapod struts to previously recorded positions.

These actuator positions must be the same as the positions prior to shutdown.

GroupReadyAtPosition(HEXAPOD,0.588,0.291,0.813,0.001,-0.140,0.970)

The example above allows the Hexapod to be at READY state at the last moved hexapod position (X, Y, Z, Tx, Ty, Tz) (1,1,1,0,0,0) after power shutdown.

Note

All other settings will remain the same, Work and Tool Coordinate Systems, for example.

WARNING

In order to avoid the possible loss of position accuracy, actual positions of the actuators must be carefully entered in the GroupReadyAtPosition() function. A standard homing process is recommended, whenever possible, after moving to a safe position.

8.4.2 Absolute Moves (HexapodMoveAbsolute)

The function **HexapodMoveAbsolute (GroupName, Work, X, Y, Z, U, V, W)** moves the Hexapod to the position (X Y Z U V W).

To be more precise, it moves the Tool coordinate system, and hence the carriage of the Hexapod that moves with the Tool coordinate system, to the position (X Y Z U V W) in the work coordinate system. This context is discussed in details in chapter 8.0 and important for a correct understanding of the Hexapod motion. For illustration, Figure 17 shows some simple examples of different absolute positions.

In the default configuration of the Hexapod, the center of tool coordinate system is in the center of the top plate of the Hexapod carriage. This is the pivot point for all rotations.

The position of the Tool coordinate system with respect to the carriage of the Hexapod, can be relocated, see chapter 9.4.6 for details.

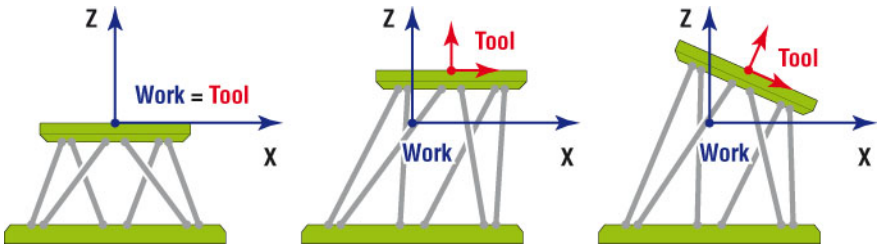


Figure 17: Default HXP configuration:
Left: Hexapod at position (0 0 0 0 0 0), Tool coincident with work
Middle:Hexapod at position (X 0 Z 0 0 0)
Right: Hexapod at position (X 0 Z 0 V 0)

All move commands are synchronized on the Hexapod positioners (the Hexapod struts). This means all positioners start and stop at the same time and have always completed the same portion of their travel distance. The “slowest” positioner defines the speed and acceleration of the other positioners. If all positioners have the same speed, the “slowest” positioner is the one that has the longest distance to travel.

NOTE

As the motion is synchronized on the Hexapod struts, the motion does not follow an ideal “straight” trajectory in the work coordinate system. For a 1 mm long trajectory, for example, this may result in a deviation of ~2 µm from the ideal trajectory in the work coordinate system.

8.4.3 Incremental Moves Along and Around Tool (HexapodMoveIncremental)

The function **HexapodMoveIncremental (GroupName, Tool, X, Y, Z, U, V, W)** performs an incremental move along and around the axis of the tool coordinate system.

Different than the absolute move, that uses the work coordinate system for reference, the incremental Tool move uses the Tool coordinate system before the motion for reference, let's call it ToolFix, and moves the tool coordinate system to the position (X Y Z U V W) in the ToolFix coordinate system.

Figure 18 shows a simple example for a linear incremental tool motion.

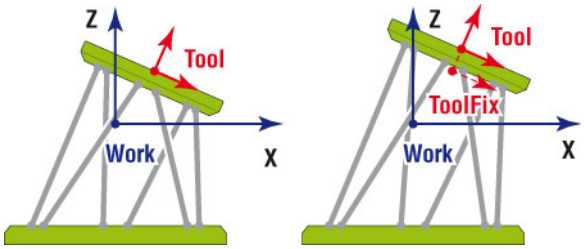


Figure 18: Left: Hexapod at position (X 0 Z 0 V 0)
Right: Hexapod after HexapodMoveIncremental (HEXAPOD, Tool, 0, 0, Z, 0, 0, 0)

Incremental rotations around tool rotate the Tool Coordinate System around the ToolFix system. The lateral positions X, Y, Z of Tool in Work don't change. This is the same as for absolute rotations. But the axes for rotations are those of the ToolFix system and not those of the Work system, see Figure 20 for illustrations.

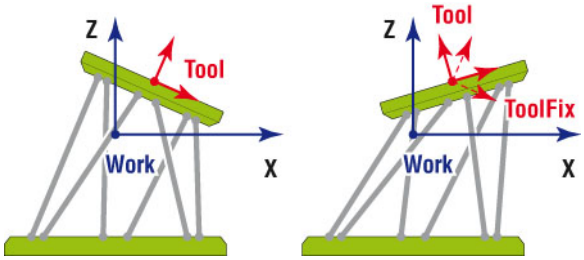


Figure 19: Left Hexapod at position (X 0 Z 0 V 0)
Right: Hexapod after HexapodMoveIncremental (HEXAPOD, Tool, 0, 0, 0, 0, V, 0)

Although this function is very well defined and can be used to increment all 6 coordinates at the same time, it is recommended to increment either only linear coordinates (XYZ), or only one angular coordinate (U, V, or W) at a time.

8.4.4 Incremental Moves Along and Around Work (HexapodMoveIncremental)

The function **HexapodMoveIncremental (GroupName, Work, X, Y, Z, U, V, W)** performs an incremental move along and around the axis of the work coordinate system.

Different than the incremental move along and around tool, the incremental move along and around work uses the work coordinate system as reference.

When used for linear incremental motion with no incremental rotation, the final XYZ position is equal to the XYZ position before the motion, plus the increment:

Example

GroupPositionCurrentGet (HEXAPOD)

Returns the current position of HEXAPOD. The return is: X, Y, Z, U, V, W

HexapodMoveIncremental (HEXAPOD, Work, x, y, z, 0, 0, 0)

Performs an incremental move along and around work. When done,

GroupPositionCurrentGet (HEXAPOD)

returns: X+x, Y+y, Z+z, U, V, W

Incremental rotations around Work rotate the Tool Coordinate System **around** the Work coordinate system. So different than absolute moves or incremental moves around Tool, an incremental rotation around Work changes the XYZ position of the Hexapod, unless the Hexapod is at X=Y=Z=0. Figure 20 shows an example of an incremental Work rotation for illustration.

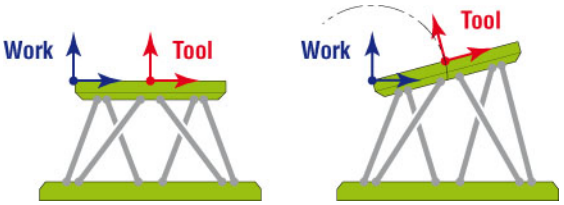


Figure 20: Left: Hexapod at position (X Y Z U V W)
Right: Hexapod after HexapodMoveIncremental (HEXAPOD, Work, 0, 0, 0, 0, V, 0

Although this function is very well defined and can be used to increment all 6 coordinates at a time, it is recommended to increment either only linear coordinates (XYZ), or only one angular coordinate (U, V, or W) at a time.

8.4.5 Moves of the Hexapod Struts (GroupMoveAbsolute and GroupMoveRelative)

The functions **GroupMoveAbsolute (GroupName or PositionerName)** and **GroupMoveRelative (GroupName or PositionerName)** move an individual Hexapod strut (PositionerName) or all Hexapod struts (GroupName) to a desired position (absolute move) or by a defined increment (relative move).

Moving an individual Hexapod strut may result in a surprising motion of the Hexapod platform. Hence, these functions should be used carefully. The primarily use of these functions is for maintenance, only.

8.4.6 Changing the Position of the Tool and Work Coordinate Systems

The default positions of the Work coordinate system (in World) and the Tool coordinate system (in Carriage) are set in the system.ini configuration file of the HXP controller, see chapter 8.0 for details. Both systems positions can be changed with the function **HexapodCoordinateSystemSet (GroupName, Tool or Work, X, Y, Z, U, V, W)**. The function **HexapodCoordinateSystemGet (GroupName, Tool or Work, X, Y, Z., U, V, W)** returns the current position of the Tool or work system.

The function HexapodCoorindateSystemSet () does not change the settings in the system.ini file. It only changes the current positions of Tool and Work. After booting the HXP controller, always the default values from the system.ini are used.

In the default configuration of the HXP, the center of the Tool coordinate system is at the center of the upper surface of the Hexapod carriage, Z is orthogonal to the carriage, and X is in the direction of the motor cables, when the Hexapod is at its reference Home position.

This means the default pivot point for all absolute moves (HexapodMoveAbsolute ()) and for all incremental rotations around Tool (HexapodMoveIncremental (GroupName, Tool)) is the center of the top surface of the carriage. To change this pivot point, one must change the position of the Tool coordinate system.

Example

The HXP is in the default configuration. After homing (state 11), the function

GroupPositionCurrentGet (HEXAPOD)

returns: 0, 0, 0, 0, 0, 0. The function

HexapodCoordinateSystemGet (HEXAPOD, Tool)

returns the current position of the Tool coordinate system (in carriage). The return is: 0, 0, 25, 0, 0, 0. The function

HexapodCoordinateSystemSet (HEXAPOD, Tool, 0, 0, 75, 0, 0, 0)

sets the pivot point for rotations 50 mm above the center of top surface of the Hexapod carriage. The function

GroupPositionCurrentGet (HEXAPOD)

now returns: 0, 0, **50**, 0, 0, 0. This is because changing the position of the Tool coordinate system in the Carriage system also changes the position of the Tool coordinate system in the Work system.

Let's assume the Home position of HEXAPOD is when all Hexapod struts are close to their lower travel limit. This is true for most of Newport's Hexapods. In this case, the position 0, 0, 50, 0, 0, 0 is the "lowest" position that the Hexapod can reach (see example in chapter 0 for further explanations). To move the Hexapod to its "optimum working position" one would need to move the Hexapod to its mid-point z-travel position.

Example

Starting from the example above, the function

HexapodMoveAbsolute (HEXAPOD, Work, 0, 0, 64, 0, 0, 0),

Moves the Hexapod to its "optimum working position", assuming that the Hexapod has a Z travel range of 28 mm, see documentation of the Hexapod mechanics for details.

So the position (0, 0, 64, 0, 0, 0) is the "optimum working position" with a Tool coordinate system that is 50 mm above the center of the carriage. This position may be uncomfortable, particular if the application can benefit also from incremental rotations around Work. Please remember, incremental rotations around Work perform a rotation around the axis of the Work system and in this case an X and Y axis that is 64 mm "underneath" the "optimum working position". The result is a concave motion with a radius of 64 mm to the center of Tool (that is 50 mm above the carriage). This may or may not be wanted. To change this behaviour one can change the position of the Work system.

Example

Starting from the example above, the function

HexapodCoordinateSystemGet (HEXAPOD, Work)

returns the current position of the Work coordinate system (in world). The return is: 0, 0, 196, 0, 0, 0. The function

HexapodCoordinateSystemSet (HEXAPOD, Work, 0, 0, 260, 0, 0, 0)

changes the position of the Work coordinate system. The function

GroupPositionCurrentGet (HEXAPOD)

now returns: 0, 0, **0**, 0, 0, 0. This is because changing the position of the Work coordinate system also changes the position of the Tool coordinate system in the Work system. Now Tool equals Work at the "optimum working position" of the Hexapod.

If these changes shall be done permanent, it is recommended to change the positions of the Tool and the Work coordinate systems in the HXP system.ini file and to launch the function **HexapodMoveAbsolute (HEXAPOD, Work, 0, 0, 0, 0, 0, 0)** after every **GroupHomeSearch (HEXAPOD).**

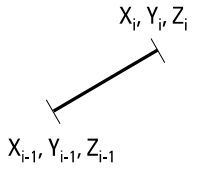
8.4.7 RightPath™ Trajectories

The HXP controller RightPath™ features 3 different types of trajectories along and around the Work or the Tool coordinate system strictly for the hexapod:

The Line trajectory is a trajectory defined by a straight line segment.

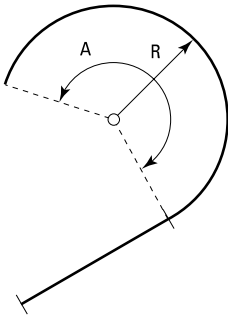
A line element is defined by specifying the (X, Y, Z) ending point in the Work or Tool coordinate system.

All line element positions are defined relative to the trajectory's starting point.

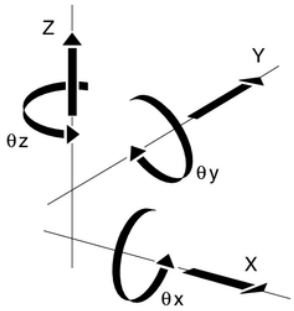


The Arc trajectory is a trajectory defined by a curve segment.

An arc is defined by specifying the radius R, the sweep angle A and the rotation Axis.



The rotation trajectory is a trajectory defined by a rotation around X, Y or Z axis.



These trajectories are available only for HEXAPOD group. The major benefit of these trajectories is the ability to maintain constant speed (speed being the scalar of the trajectory velocity) throughout the entire path, excluding the acceleration and deceleration periods. The trajectory is user defined by API's described in this section. Once the user executes the API function to begin the trajectory, the HXP automatically calculates the needed moves for all Hexapod struts and executes the motion in the Work or the Tool coordinate system. A dedicated function performs a precheck of the trajectory which returns the maximum and minimum travel requirement per each Hexapod strut as well as maximum possible trajectory speed and trajectory acceleration that is compatible with the different positioner parameters.

The function **HexapodMoveIncrementalControl** executes Line, Arc or Rotation trajectory with the maximum velocity.

The function **HexapodMoveIncrementalControlWithTargetVelocity** executes Line, Arc or Rotation trajectory with the specified velocity.

8.4.7.1 Trajectory Terminology

Trajectory: defined as a continuous multidimensional motion path. Hexapod Line, arc or rotation trajectories are defined in a two-dimensional plane within the Tool or Work Coordinate systems. These are used with only Hexapod groups. The main requirement

of a line, arc or rotation is to maintain a constant speed (speed being the scalar of the vector velocity) throughout the entire path (except during the acceleration and deceleration periods).

Trajectory velocity: the tangential linear velocity (speed) along the trajectory during its execution

Trajectory acceleration: the tangential linear acceleration used to start and end a trajectory. Trajectory acceleration and trajectory deceleration are equal by default.

Lag time: Prior to executing the trajectory with **HexapodMoveIncrementalControl()**, the HXP controller calculates the needed move for all Hexapod struts, and the time it takes to process this calculation is called Lag time. The lag time depends on the number of points in trajectory, which is determined by the **ProfileGeneratorISRRatio** parameter in "system.ref" file.

The Profile Generator loop calculates the setpoints in a trajectory at a rate relative to the Servo rate (or CorrectorISRPeriod). The default ratio is for HXP controller is set at twenty CorrectorISRPeriods to one Profile Generator loop hence the default ProfileGeneratorISRRatio is 20.

ProfileGeneratorISRRatio = Servo Rate / Profile Generator Rate

Servo Rate = (CorrectorISRPeriod)-1

For the HXP controller, the Profile Generator Rate is up to 2.5 kHz while the Servo Rate is adjustable up to 10 kHz.

The number of points on the trajectory is limited to 50,000.

The time interval between two points (Profiler Period) is equal to CorrectorISRperiod * ProfileGeneratorISRRatio

Typical values of these parameters for an Hexapod Group are:

CorrectorISRperiod = 160 µs

ProfileGeneratorISRRatio = 20

The maximum duration of a trajectory in this case is 50,000 * 160 µs * 20 = 160 s.

The minimum jerk time must be > 2 * CorrectorISRperiod * ProfileGeneratorISRRatio = 0.005 s.

8.4.7.2 Trajectory Conventions

When defining and executing a line, arc or rotation trajectory, a number of rules must be followed:

- The motion group must be a HEXAPOD group.
- A Line trajectory is available in the Tool or Work coordinate systems. The displacement value for a Line trajectory is not limited.
- An Arc trajectory is available in the Work coordinate system. The distance between Tool and Work defines the radius of the Arc. When the arc is defined with 3 axis, the displacement value is between - 90º and + 90º (the limitation is due to the Bryant angle representation). When the arc is defined with only one axis, a value greater than 90 º can be used.
- A Rotation trajectory is available in the Work coordinate system. The displacement value is between - 90º and + 90º (the limitation is due to the Bryant angle representation).

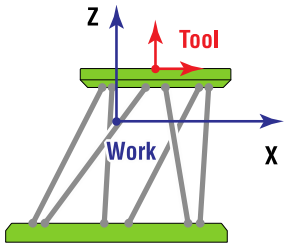


Figure 21: RightPath™ Trajectory Conventions in HXP default coordinate systems — Line Trajectory is defined in Tool or Work coordinate systems; Arc or Rotation trajectory is defined in Work coordinate system only.

8.4.7.3 Geometric Conventions

The coordinate system of a Line trajectory is the Tool or Work coordinate system. The coordinate system of an Arc or Rotation trajectory is the Work coordinate system.

For the details of Hexapod Tool and Work coordinate systems and Bryant angle definition, refer to chapter 8.0 Coordinate Systems.

8.4.7.4 Trajectory Verification and Execution

There are 5 functions to verify or execute a trajectory:

- **HexapodMoveIncrementalControl(GroupName, CoordinateSystem, HexapodTrajectoryType, dX, dY, dZ):** Executes line, arc or rotation trajectory with the maximum velocity
- **HexapodMoveIncrementalControlwithTargetVelocity (GroupName, CoordinateSystem, HexapodTrajectoryType, dX, dY, dZ, Velocity):** Executes line, arc or rotation trajectory with the specified velocity
- **HexapodMoveIncrementalControlLimitGet (GroupName, CoordinateSystem, HexapodTrajectoryType, dX, dY, dZ):** Returns the maximum velocity and the percent of trajectory executable versus the target distance entered
- **HexapodMoveIncrementalPulseAndGatheringSet (GroupName, Divisor):** Configures gathering data and generates pulses on GPIO2 (Pin 11 and 12) during only a constant velocity
- **HexapodSGammaParametersDistanceGet (PositionerName, Displacement, Velocity, Acceleration, MinimumJerkTime, MaximumJerkTime):** Returns the distance during acceleration phase and distance during constant velocity phase in the virtual SGamma profiler used to define a Hexapod trajectory

8.4.7.5 Line Trajectory

The function **HexapodMoveIncrementalControl (GroupName, Work, Line, X, Y, Z)** performs a line trajectory move along and around the XYZ axis of the work coordinate system.

The function **HexapodMoveIncrementalControlWithTargetVelocity(GroupName, Work, Line, X, Y, Z, A)** performs a line trajectory move along and around the XYZ axis of the work coordinate system with target velocity of A mm/s.

Example 1

Work: Line from [0, 0, 0] to [17, 0, 0]

HexapodMoveAbsolute(HEXAPOD, Work, 0, 0, 0, 0, 0, 0)

Moves the carriage to zero position

HexapodMoveIncrementalControlLimitGet(HEXAPOD, Work, Line, 100, 0, 0, double*, double*)

The return is: 0, 5.921847271479, 0.17222222222222

HexapodMoveIncrementalControlLimitGet(Hexapod, Work, Line, 17, 0, 0, double*, double*)

The return is: 0, 10.08298747472,1

HexapodMoveIncrementalControl(Hexapod, Work, Line, 17, 0, 0)

Performs a line trajectory of 17 mm move along X axis

In this example, displacement of 100 units is not an allowed motion as it exceeds the maximum travel range available in the X axis. The **HexapodMoveIncrementalControlLimitGet** function does the verification before performing the line trajectory move. When 100 units in X are entered, the **HexapodMoveIncrementalControlLimitGet** returns 0.1722222. This indicates that the maximum travel range in X axis is 100 * 0.1722222 = 17.22222. When the 17 units in X are entered, the **HexapodMoveIncrementalControlLimitGet** returns the maximum velocity of 10.08 unit/s. This indicates that the maximum Hexapod velocity in X direction is 10.08 mm/s.

Note

HexapodMoveIncrementalControlLimitGet returns the correct maximum velocity value only when the percent of trajectory executable is equal to 1.

Example 2

Work: Line from [0, 0, 0] to [17, 0, 0] with 10 mm/s

HexapodMoveAbsolute(HEXAPOD, Work, 0, 0, 0, 0, 0, 0)

Moves the carriage to zero position

HexapodMoveIncrementalControlWithTargetVelocity(Hexapod, Work, Line, 17, 0, 0, 10)

Performs a line trajectory of 17 mm move along X axis with 10 mm/s velocity

8.4.7.6 Arc Trajectory

The function **HexapodMoveIncrementalControl(GroupName, Work, Arc, X, Y, Z)** performs an arc trajectory move in the work coordinate system.

The function **HexapodMoveIncrementalControlWith TargetVelocity(GroupName, Work, Arc, X, Y, Z, A)** performs an arc trajectory move in the work coordinate system with target velocity of A °/s.

The arc trajectory move is available in the work coordinate system. The distance between Tool and Work defines the radius of the arc.

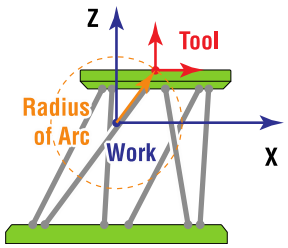


Figure 22: RightPath™ Arc Trajectory — The distance between Tool and Work coordinates defines the radius of the arc.

Example 1

Arc movement from [0, 0, 4] with 4 mm radius from 0 to 100 degree

HexapodMoveAbsolute(HEXAPOD, Work, 0, 0, 4, 0, 0, 0)

Moves the carriage to 0, 0, 4, 0, 0 0 position

HexapodMoveIncrementalControlLimitGet(Hexapod, Work, Arc, 100, 0, 0, double*,double*)

The return is: 0, 54.42877230317, 1

HexapodMoveIncrementalControl (Hexapod, Work, Arc, 100, 0, 0)

Performs an arc trajectory around X axis with 4 mm radius from 0 to 100 degree angle

Example 2

Arc movement from [0, 0, 4] with 4 mm radius from 0 to 100 degree with 10 deg/s

HexapodMoveAbsolute(HEXAPOD, Work, 0, 0, 4, 0, 0, 0)

Moves the carriage to 0, 0, 4, 0, 0 0 position

HexapodMoveIncrementalControlWithTargetVelocity (Hexapod, Work, Arc, 100, 0, 0, 10)

Performs an arc trajectory around X axis with 4 mm radius from 0 to 100 degree with 10 deg/s velocity

8.4.7.7 Rotation Trajectory

The function **HexapodMoveIncrementalControl(GroupName, Work, Rotation, X, Y, Z)** performs a rotation trajectory move in the work coordinate system.

The function **HexapodMoveIncremental ControlWithTargetVelocity(GroupName, Work, Rotation, X, Y, Z, A)** performs a rotation trajectory move in the work coordinate system with target velocity of A °/s.

Example

Rotation movement from [0, 0, 0] with 8.8 degree around X axis

HexapodMoveAbsolute(HEXAPOD, Work, 0, 0, 0, 0, 0, 0)

Moves the carriage to 0, 0, 0, 0, 0 0 position

HexapodMoveIncrementalControlLimitGet(HEXAPOD, Work, Rotation, 100, 0, 0, double*,double*)

The return is: 0, 5.334928580185, 0.0888888888889

HexapodMoveIncrementalControlLimitGet (HEXAPOD, Work, Rotation, 8.88, 0, 0, double*,double*)

The return is: 0, 5.737548049716, 1

HexapodMoveIncrementalControlWithTargetVelocity (HEXAPOD, Work, Rotation, 8.88, 0, 0, 5.73)

Performs a rotation trajectory movement around X axis from 0 to 8.88 degree with 5.73 deg/s velocity

8.4.7.8 Pulse and Gathering

HexapodMoveIncrementalControlPulseAndGatheringSet(GroupName,Divisor) configures a gathering and generates pulse on GPIO2 (pin 11 and 12). Gathered data are X, Y, Z U, V, W. Pulses will be generated at the same time the data is gathered, during the constant velocity phase only.

The time interval between 2 pulses is calculated by:

Divisor * CorrectorISRPeriod

At the end of the displacement, **StopAndSaveGathering()** can be executed and the gathered data can be verified to show at which position a pulse has been generated.

Example

HexapodMoveAbsolute(HEXAPOD,Work,0,0,0,0,0,0)

Moves the carriage to zero position

HexapodMoveIncrementalControlPulseAndGatheringSet(HEXAPOD,10000)

Configures gathering with divisor 10000

HexapodMoveIncrementalControlWithTargetVelocity(HEXAPOD,Work,Line,17,0,0,10)

Performs a line trajectory of 17 mm move along X axis with 10 mm/s velocity

EventExtendedConfigurationActionGet(Char*)

The return is: 0, GatheringOneData;;;;

EventExtendedConfigurationTriggerGet(Char*)

The return is: 0, Always;;;;;Hexapod.PVT.TrajectoryPulse;;;;

GatheringConfigurationGet(Char*)

The return is:

0,HEXAPOD.X.CurrentPosition;Hexapod.Y.CurrentPosition;HEXAPOD.Z.CurrentPosition;HEXAPOD.U.CurrentPosition;Hexapod.V.CurrentPosition;Hexapod.W.CurrentPosition

GatheringStopAndSave()

In the default setting of HXP, the gathering is done at every CorrectorISRPeriod. The divisor of **HexapodMoveIncrementalControlPulseAndGatheringSet** in above example is set to 10000. This results in the sampling time of CorrectorISRPeriods.

8.5 SingleAxis Motion

The predefined home search processes described in the previous section might not be compatible with all motion devices or might not be always executable. For instance, if there is a risk of collision during a standard home search process. In other situations, a home search process might not be desirable. For example, to ensure that the stages have not moved, the current positions are stored into memory. In this case, it is sufficient to reinitialize the system by setting the position counters to the stored position values.

For these special situations, the HXP controller's referencing state is an alternative to the predefined home search processes with a GroupReferencing set of commands for the Single Axis Group and the GroupReadyAtPosition for the Hexapod Group.

NOTE

The Referencing state should be only used by experienced users. Incorrect use could cause equipment damage.

8.5.1 SingleAxis Referencing State: GroupReferencing

The Referencing state is a parallel state to the homing state, see the state diagram on page **Erreur ! Signet non défini.**, Figure 17. To enter the referencing state, send the function **GroupReferencingStart(GroupName)** while the group is in the NOT REFERENCED state.

In the Referencing state, the function **GroupReferencingActionExecute(PositionerName, Action, Sensor, Parameter)** will perform certain actions like moves, position latches of reference signal transitions, or position resets. The function **PositionerSGammaParametersSet(PositionerName)** can be used to change the velocity, acceleration and jerk time parameters.

To leave the referencing state, send the function **GroupReferencingStop(GroupName)**. The Group will then be in the HOMED state, state number 11.

The syntax and function of the function **GroupReferencingActionExecute(PositionerName, Action, Sensor, Parameter)** will be discussed in detail. With this function, there are four parameters to specify:

- PositionerName is the name of the positioner on which this function is executed.
- Action is the type of action that is executed. There are eight actions that can be distinguished into three categories: Moves that stop on a sensor event, moves of certain displacement, and position counter reset categories.
- Sensor is the sensor used for those actions that stop on a sensor event. It can be MechanicalZero, MinusEndOfRun, or None.
- Parameter is either a position or velocity value and provides further input to the function.

The following table summarizes all possible configurations:

	Sensor			Parameter	
Action	**MechanicalZero**	**MinusEndOfRun**	**None**	**Position**	**Velocity**
LatchOnLowToHighTransition	■	■			■
LatchOnHighToLowTransition	■	■			■
LatchOnIndex			■		■
LatchOnIndexAfterSensorHighToLowTransition	■	■			■
SetPosition			■	■	
SetPositionToHomePreset			■		
MoveToPreviouslyLatchedPosition			■		■
MoveRelative			■	■	

8.5.2 Move on Sensor Events

The “move on sensor events” starts a motion at a defined velocity, latches the position when a state transition of a certain sensor is detected, then stops the motion. There are four possible actions under this category:

- **LatchOnLowToHighTransition**
- **LatchOnHighToLowTransition**
- **LatchOnIndex**
- **LatchOnIndexAfterSensorHighToLow**

With **LatchOnLowToHighTransition** and **LatchOnHighToLowTransition**, latching happens when the right transition on the defined sensor occurs. The sensor can be latched to either **MechanicalZero**, **MinusEndOfRun** and **PositiveEndOfRun** when supported by the hardware, refer to §7.2 to know which hardware supports the function. With **LatchOnIndex** and **LatchOnIndexAfterSensorHighToLow**, latching happens on the index signal. With **LatchOnIndexAfterSensorHighToLow,** latching happens on the first index after a high to low transition at the defined sensor (**MechanicalZero** or **MinusEndOfRun**). Because of the dedicated hardware circuits used for the position latch, there is essentially no latency between sensor transition detection and position acquisition.

In all cases, motion stops after the latch. However, this means that the stopped position doesn’t rest on the sensor transition, but at some short distance from it. To move exactly to the position of the sensor transition, use the action **MoveToPreviouslyLatchedPosition**.

The latch does not change the current position value. In order to set the current position value, use the action **SetPosition** or **SetPositionToHomePreset**, for instance, after a **MoveToPreviouslyLatchedPosition**.

In the Referencing state, the limit switch safeties (emergency stop) are still enabled until the **MinusEndOfRun** sensor is specified with a **GroupReferencingActionExecute()** function. When specified, the limit switch safeties are disabled and will only be re-enabled with the function **GroupReferencingStop()**.

The Parameter has a sign, if it is assigned as velocity (floating point). This means that the direction of motion is dictated by the sign of the velocity parameter.

8.5.3 Moves of Certain Displacements

These two move commands which don't use the same parameters, are explained below.

- **MoveRelative**

  The action **MoveRelative** commands a relative move of a positioner similar to the function **GroupMoveRelative**. However, the function **GroupMoveRelative** is not available in the Referencing state. The relative move is specified by a positive or negative displacement. The move is done with the SGamma profiler. The speed and acceleration are the default values, or the last value defined by either a move on sensor event, a **MoveToPreviouslyLatchedPosition**, or a **PositionerSGammaParametersSet**.

- **MoveToPreviouslyLatchedPosition**

  This action moves the positioner to the last latched position, see section 9.4.1: "Move on sensor events" for details. It verifies there was a position latched since this last **GroupReferencingStart** call. This is important because an old latched position can still be in memory from a previous home search or referencing. And moving to this previous latched position could have unexpected results. The move is done with the SGamma profiler. The speed is specified by a parameter. The acceleration is the default value, or the last value defined by a **PositionerSGammaParametersSet**.

8.5.4 Position Counter Resets

"Position counter resets" sets the current position to a certain value. There are two options: **SetPosition** and **SetPositionToHomePreset**. The main use of these actions is when the positioner is at a well defined reference position after a **MoveToPreviouslyLatchedPosition** action.

Another use of this action is for a "soft" system start by Referencing a group to a known set position, without executing a home search process, for example. In this case, a suggested sequence of functions follows:

GroupReferencingStart(GroupName)

GroupReferencingActionExecute(PositionerName, "SetPosition", "None", KnownCurrentPosition)

GroupReferencingStop(GroupName)

SetPosition sets the current position to a value defined by a parameter. **SetPositionToHomePreset** sets the current position to the **HomePreset** value stored in the stages.ini configuration file. It is equivalent to a **SetPosition** of the same positioner to the **HomePreset** value.

It is important that all positioners of a motion group are referenced to a position using the **SetPosition** or **SetPositionToHomePreset** before leaving the Referencing state (see example on page 94).

8.5.5 Example: MechanicalZeroAndIndexHomeSearch

The following sequence of functions has the same effect as the MechanicalZeroAndIndexHomeSearch:

```
GroupReferencingStart(GroupName)
PositionerHardwareStatusGet (PositionerName, &status)
if ((status & 4) == 0) { // 4 is the Mechanical zero mask on the hardware status
GroupReferencingActionExecute(PositionerName, "LatchOnLowToHighTransition",
"MechanicalZero, -10) }
GroupReferencingActionExecute(PositionerName, "LatchOnHighToLowTransition",
"MechanicalZero", 10)
```

```
GroupReferencingActionExecute(PositionerName, “LatchOnLowToHighTransition”, “MechanicalZero”, -5)
GroupReferencingActionExecute(PositionerName, “LatchOnIndexAfterSensorHighToLow”, “MechanicalZero”, 5)
GroupReferencingActionExecute(PositionerName, “MoveToPreviouslyLatchedPosition”, “None”, 5)
GroupReferencingActionExecute(PositionerName, “SetPositionToHomepreset”, “None”, 0)
GroupReferencingStop(GroupName)
```

8.5.6 SingleAxis Move

The HXP controller supports two methods to define when a motion is completed (MotionDone): The “theoretical MotionDone” and the “VelocityAndPositionWindow MotionDone”. The preferred method for MotionDone used is set up in the stages.ini file. In theory, MotionDone is when the motion is completed by the profiler. It does not take into account the settling of the positioner at the end of the move. So depending on the precision and stability requirements at the end of the move, the theoretical MotionDone might not always be the same as the physical end of the motion. The VelocityAndPositonWindow MotionDone allows a more precise definition by specifying the end of the move with a number of parameters that take the settling of the positioner into account. In the VelocityAndPositionWindow MotionDone the motion is completed when:

> | PositionErrorMeanValue | < | MotionDonePositionThreshold | AND | VelocityMeanValue | < | MotionDoneVelocityThreshold | is verified during the MotionDoneCheckingTime period.

The different parameters have the following meaning:

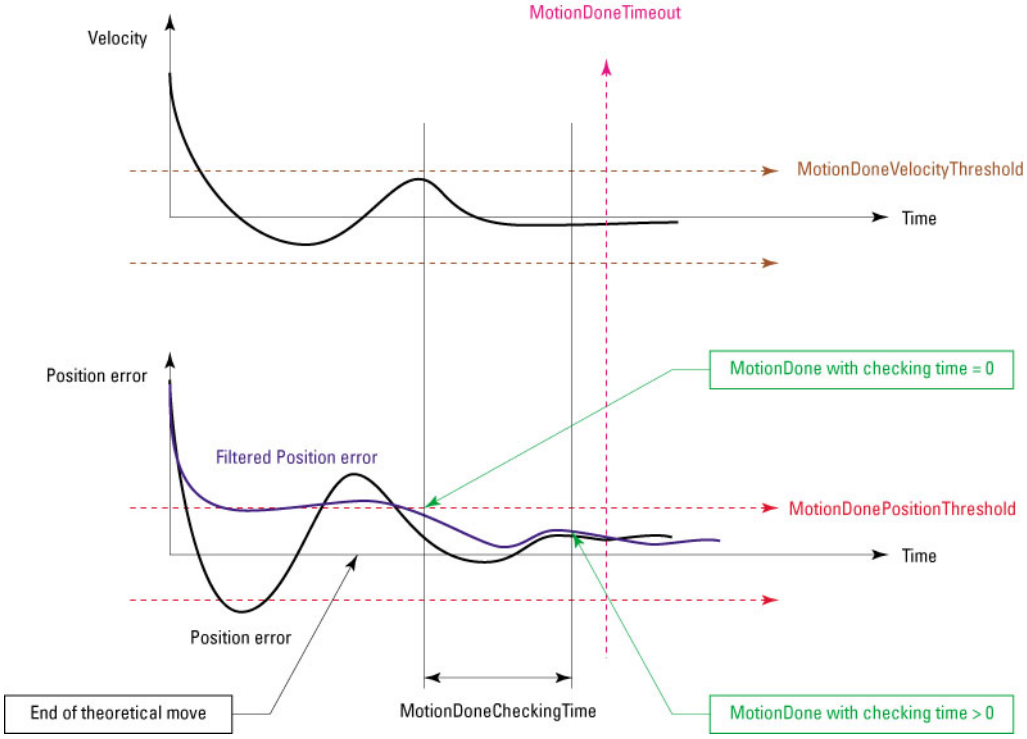


Figure 23: Motion Done.

- **MotionDonePositionThreshold**: This parameter defines the position error window. The position error has to be within ± of this value for a period of MotionDoneCheckingTime to validate this condition.
- **MotionDoneVelocityThreshold**: This parameter defines the velocity window. The velocity at the end of the motion has to be within ± of this value for a period of MotionDoneCheckingTime to validate this condition.

- **MotionDoneCheckingTime**: This parameter defines the period during which the conditions for the MotionDonePositionThreshold and the MotionDoneVelocityThreshold must be true before setting the motion done.
- **MotionDoneMeanPeriod**: A sliding mean filter is used to attenuate the noise for the position and velocity parameters. The MotionDoneMeanPeriod defines the duration for calculating the sliding mean position and velocity. The mean position and velocity values are compared to the threshold values as defined above. This parameter is not illustrated on the graph.
- **MotionDoneTimeout**: This parameter defines the maximum time the controller will wait from the end of the theoretical move for the MotionDone condition, before sending a MotionDone time-out.

Important

The HXP controller can only execute a new move on the same positioner or on the same motion group when the previous move is completed (MotionDone) and when the positioner or the motion group is again in the ready state.

The HXP controller can trigger an action when the motion is completed (MotionDone) or when the motion is ended (theoretical motion end by the profiler) by using the events MotionDone or MotionEnd. For further details see chapter 12.0.

The functions **PositionerMotionDoneGet()** and **PositionerMotionDoneSet()** allow reading and modifying the parameters for the VelocityAndPositionWindow MotionDone. These parameters are only taken into account when the MotionDoneMode is set to VelocityAndPositionWindow in the stages.ini.

Example

Modifications of the MotionDoneMode can be made only manually in the stages.ini file. The stages.ini file is located in the config folder of the HXP controller, see chapter 5: “FTP (File Transfer Protocol) Connection” for details.

Make a copy of the stages.ini file to the PC. Open the file with any text editor and modify the MotionDoneMode parameter of appropriate stage to VelocityAndPositionWindow, and set the following parameters:

```
;--- Motion done
MotionDoneMode = VelocityAndPositionWindow  ; instead of Theoretical
MotionDonePositionThreshold = 4             ; units
MotionDoneVelocityThreshold = 100           ; units/s
MotionDoneCheckingTime = 0.1                ; seconds
MotionDoneMeanPeriod = 0.001                ; seconds
MotionDoneTimeout = 0.5                     ; seconds
```

Replace the current stages.ini file on the HXP controller with this modified version (make a copy of the old .ini file first). Reboot the controller. To apply any changes to the stages.ini or system.ini, the controller has to reboot.

Use the following functions:

GroupInitialize (HEXAPOD)

GroupHomeSearch (HEXAPOD)

PositionerMotionDoneGet (HEXAPOD.1)

This function returns the parameters for the VelocityAndPositionWindow Motion done previously set in the stages.ini file, so 4, 100, 0.1, 0.00 and 0.5.

PositionerMotionDoneSet (HEXAPOD.1, PositionThresholdNewValue, VelocityThresholdNewValue, CheckingTimeNewValue, MeanPeriodNewValue, TimeoutNewValue)

This function replaces the parameters by the new entered values.

8.5.7 Master Slave

In master slave mode, any motion axis can be electronically geared to another motion axes, or a single master with multiple slaves. The gear ratio between the master and the slave is user defined. During motion, all axes compensations of the master and the slave are taken into account.

The slave must be a SingleAxis group. The master can be a positioner from any group. The Master slave relation is set by the function **SingleAxisSlaveParametersSet().**

The Master slave mode is enabled by the function **SingleAxisSlaveModeEnable()**. To enable the Master slave mode, the Slave group must be in the ready state. The Master group can be in the not-referenced or ready state.

Example 1

This example shows the sequence of functions used to set-up a master-slave relation between two axes that are not mechanically joined (meaning the two axis can move independently):

GroupInitialize (SlaveGroup)

GroupHomeSearch (SlaveGroup)

GroupInitialize (MasterGroup)

GroupHomeSearch (MasterGroup)

…

SingleAxisSlaveParametersSet (SlaveGroup, MasterGroup.Positioner, Ratio)

SingleAxisSlaveModeEnable (SlaveGroup)

GroupMoveRelative (MasterGroup.Positioner, Displacement)

…

SingleAxisSlaveModeDisable (SlaveGroup)

Example 2

This example shows the sequence of functions used to set-up a Master slave relation **between two axes that are mechanically joined**. Different from example 1, all motions, including the motion done during the home search routine, are performed synchronously.

Important: First, set the HomeSearchSequenceType of the Slave group's positioner to CurrentPositionAsHome in the stages.ini and reboot the HXP controller.

GroupInitialize (SlaveGroup)

GroupHomeSearch (SlaveGroup)

GroupInitialize (MasterGroup)

SingleAxisSlaveParametersSet (SlaveGroup, MasterGroup.Positioner, Ratio)

SingleAxisSlaveModeEnable (SlaveGroup)

GroupHomeSearch (MasterGroup)

…

GroupMoveRelative (MasterGroup.Positioner, Displacement)

NOTE

The slave positioners should have similar capabilities as the master positioner in terms of velocity and acceleration. Otherwise the full capabilities of the master or the slave positioners may not be utilized.

8.6 Position Information

The HXP uses 4 types of position information: TargetPosition, SetpointPosition, FollowingError and CurrentPosition. The meaning of these is as follow:

- The CurrentPosition is the current physical position of the positioner. It is equal to the encoder position after all compensations (backlash, linear error, hysteresis and mapping) have been taken into account.
- The SetpointPosition is the theoretical position commanded to the servo loop. It is the position where the positioner should be at this moment in time. The SetpointPositions are calculated by the profiler of the HXP controller in relation to the motion command and the motion parameters (speed, acceleration, jerk, etc.).
- The FollowingError is the difference between the CurrentPosition and the SetpointPosition.
- The TargetPosition is the position where the positioner must be after the completion of a move.

The CurrentPosition, SetpointPosition, and TargetPosition can be queried from the controller using the appropriate function calls. The behavior of all three functions is similar:

The function **GroupPositionCurrentGet (HEXAPOD)** returns the X, Y, Z, U, V, and W coordinates of the Hexapod platform (the position of the Tool coordinate system in the work coordinate system).

The function **GroupPositionCurrentGet (HEXAPOD.X)** returns only the X coordinate of the Hexapod platform. The same behavior for HEXPAPOD.Y for the Y coordinate, HEXAPOD.Z for the Z coordinate, HEXAPOD.U for the U coordinate, etc. The coordinate names HEXAPOD.X, etc. can be also used for data gathering, see chapter13.0.

The function **GroupPositionCurrentGet (HEXAPOD.1)** returns the length of the Hexapod strut 1. The same behavior for HEXPAPOD.2 for the second strut, HEXAPOD.3 for the third strut, etc. The position of the Hexapod struts is primarily of interest for maintenance, only.

9.0 Error Compensation

The HXP controller features different compensation methods that can improve the performance of a motion system:

9.1 Backlash Compensation

Backlash compensation improves the bi-directional repeatability and bi-directional accuracy of a motion device that has mechanical play. When the backlash compensation is activated, the HXP controller adds a user-defined BacklashValue to the internally used TargetPosition whenever reversing the direction of motion. This only internally used new target position is then the basis for the calculations of the motion profiler. This compensation is fully transparent for the user. This means the CurrentPosition, SetpointPosition and TargetPosition that can be queried from the HXP don't include this backlash value.

The backlash compensation works only under certain conditions:

- The "HomeSearchSequenceType" in the stages.ini must be different than "CurrentPositionAsHome".
- Backlash compensation is not compatible with positioner mapping, see chapter 11.4. So for positioners with backlash compensation it is not allowed to have any entry for "PositionerMappingFileName" in the stages.ini configuration file.
- There must be a value larger than zero for "backlash" in the stages.ini configuration file.

The backlash compensation is activated at power up additional functions to disable / enable the backlash are provided, for further information please refer to the programmer's manual.

The hysteresis and backlash compensations are exclusive, only one at a time can be different from zero in the "stages.ini" file.

NOTE

With Newport Hexapods that have backlash in the Hexapod actuators, the backlash is individually measured and specified in the stages.ini configuration file.

For most applications, Newport recommends using these backlash values and enabling the backlash compensation.

Example

In the stages.ini file are individual values provided for the backlash of all 6 Hexapod actuators. Example:

```
;--- Backlash
Backlash = 0.006453                    ; units
```

9.2 Hysteresis Compensation

When hysteresis compensation is set to a value different than zero, the HXP will issue for each move in the positive direction a move of the commanded distance plus the hysteresis compensation value, and then at the end of this first move a second move of the hysteresis compensation value in the negative direction. This motion ensures that a

final position gets always approached from the same direction and helps compensating for mechanical defects. This compensation is fully automated once it is set up.

The hysteresis and backlash compensations are exclusive, only one at a time can be different from zero in the “stages.ini” file

NOTE

Incase of “Abort” or “following error” during a move, a positive move is recommended to apply the hysteresis correction.

NOTE

The positive software travel limit (MinimumTargetPosition) must allow the hysteresis extra move before reaching the electrical or mechanical end of run to avoid errors.

9.3 Linear Error Correction

The linear error compensation helps improving the accuracy of a motion device by eliminating linear error sources. Linear errors can be caused by screw pitch errors, linearly increasing angular deviations (abbe errors), thermal effects or cosine errors (angles between feedback device and direction of motion). The linear error compensation can eliminate these errors. Its value is defined in the stages.ini (search for parameter LinearEncoderCorrection). When set different than zero, the encoder positions are automatically compensated by this value as follow:

$$\text{Corrected position} = \text{HomePreset} + (\text{EncoderPosition} - \text{HomePreset}) \times (1 + \text{LinearEncoderCorrection}/10^6)$$

The LinearEncoderCorrection is specified in ppm (parts per million). The correction is applied relative to the physical home position of the positioner (the Encoder position by definition is set to the HomePreset value at the home position). This hardware reference for linear error correction has the advantage of being independent of the value for the HomePreset.

Linear error compensation can be used in parallel to a positioner mapping, see chapter 11.4. For this reason, care must be taken to the effects when using linear error compensation and positioner mapping at the same time.

NOTE

With Newport Hexapods the values for the EncoderResolution and for the LinearErrorCorrection are factory set to provide optimum results. It is not recommended changing these values from their default settings without consulting to a Newport applications specialist.

The function of the Linear Error Correction is fully transparent for the user. Average users should not be concerned by the Linear Error Correction.

9.4 Positioner Mapping

In contrast to the linear error compensation, positioner mapping allows to compensate also for nonlinear error sources. Positioner mapping is enabled by a compensation table in the HXP controller and certain settings in the stages.ini. Positioner mapping can be used in addition to Linear Error compensation, but is not compatible with backlash compensation.

Users who want to learn more about Positioner Mapping are referred to the XPS user’s manual. None of the Newport Hexapods use Positioner Mapping.

10.0 Event Triggers

The HXP event triggers are similar to IF/THEN statements in programming. “If” an **event** occurs, “then” an **action** is triggered. Programmer’s can trigger any action (from a list of possible actions, see section 12.2) at any event (from a large list of possible events, see section 12.1). It is also possible to trigger several actions at the same event. Furthermore, it is possible to link several events to an event configuration. In this case, all events have to happen at the same time to trigger the action(s), comparable to a logic AND.

Some events are punctual events like “motion start”. They will trigger an action only once when the event occurs. Some other events have duration like “motion state”. They will trigger the same action every time (as applicable) as long as the event occurs. For events with duration, the event can be also considered as a statement that is checked whether it is true or not. A third event category are the permanent events “Always” (happens always) and “Timer” (happens every n^{th} servo cycle). They will trigger at every n^{th} servo cycle.

As the HXP controller provides the utmost flexibility on programming event triggers, the user must be careful and consider possible unwanted effects. Some events might have duration although only a single action at one time is asked. Some other events might never occur. This is especially true when linking several events to an event configuration. The different possible effects get illustrated in section 0 by numerous examples.

To trigger an action at an event, first the event and the associated action need to be configured. This is done by the functions **EventExtendedConfigurationTriggerSet()** and **EventExtendedConfigurationActionSet()**. Then, the event trigger needs to get activated using the function **EventExtendedStart()**. When activated, the HXP controller checks for the event each servo cycle, or each profiler cycle for those events that are motion related, and triggers the action when the event happens. Hence, the maximum latency between the event and the action is equal to the servo cycle time of as short as 100 µs or equal to the profiler cycle time of as short as 400 µs. For events with duration, that means the same action is triggered each servo cycle, or each profiler cycle, as long as the event is happening.

Event triggers (and their associated action) get automatically removed when the event configuration is not happening anymore after the event configuration has happened at least once. The only exception is if the event configuration contains any of the permanent events “Always” or “Timer”. In that case the event trigger will always stay active. With the function **EventExtendedRemove()**, any event trigger can be removed.

The function **EventExtendedWait()** can be used to halt a process. It essentially blocks the socket until the event occurs. Once the event occurs, it gets deleted. It requires a preceding function **EventExtendedConfigurationTriggerSet()** to define the event at which the process continues.

The functions **EventExtendedGet()** and **EventExtendedAllGet()** return details of the event and action configurations.

10.1 Events

General events are defined as "**Always**", "**Immediate**" and "**Timer**". With the event "Always", an action is triggered at each servo cycle. For events that are defined as "Immediate", an action is triggered once immediately (means during the very next servo cycle). For the events "Timer" an action is triggered immediately and at every n^{th} servo cycle. Here, "n" corresponds to the "FrequencyTicks" defined with the function **TimerSet()**. There are five different timers available that get selected by the actor (1…5). (Actor is the object that actions/events are linked to)

All events that are motion related, these are all events that have the Category "Sgamma", refer to the motion profiler of the HXP controller. The motion profiler runs at a frequency of up to 2.5 kHz, or every 400 µs. Thus, events triggered by the motion profiler have a maximum resolution of 400 µs. Consequently, events with duration, such as MotionState, will trigger an action at the motion profiler rate rather than the faster servo rate.

Actor			Category	Event Name	Parameter			
Positioner	GPIO	TimerX	SGamma		1	2	3	4
				Immediate				
				Always				
		■		Timer				
■			■	MotionStart				
■			■	MotionEnd				
■			■	MotionState				
■			■	MotionDone				
■			■	ConstantVelocityStart				
■			■	ConstantVelocityEnd				
■			■	ConstantVelocityState				
■			■	ConstantAccelerationStart				
■			■	ConstantAccelerationEnd				
■			■	ConstantAccelerationState				
■			■	ConstantDecelerationStart				
■			■	ConstantDecelerationEnd				
■			■	ConstantDecelerationState				
	■			DILowHigh	Bit index			
	■			DIHighLow	Bit index			
	■			DIToggled	Bit index			
	■			ADCHighLimit	Value			
	■			ADCLowLimit	Value			
■				PositionerError	Mask			
■				PositionerHardwareStatus	Mask			

An event is entirely composed by:

[Actor].[Category].Event Name, Parameter1, Parameter2, Parameter3, Parameter4

Not all event names have a preceding actor and category, but all events have four parameters, even though most parameters are not used today. For all unused parameters with, it is still required to have a zero (0) as default.

Please note, the separator between the category, the actor, and the event name is a dot (.). The separator between the Event Name and the parameters is a comma (,).

To define an Event, use the function EventExtendedConfigurationTriggerSet().

Examples

EventExtendedConfigurationTriggerSet (HEXAPOD.1.SGamma.MotionStart, 0, 0, 0, 0)

In this case the actor is a positioner (HEXAPOD.1) and the event has a category. The event happens when the next motion on the positioner HEXAPOD.1 starts. After the motion is started, the event gets removed.

EventExtendedConfigurationTriggerSet (GPIO2.ADC2.ADCHighLimit, 3, 0, 0, 0)

In this case the actor is a GPIO name (GPIO2.ADC2) and the event has no category. The event happens when the voltage on the GPIO.ADC2 exceeds 3 Volts.

It is also possible to link different events to an event configuration. The same function EventExtendedConfigurationTriggerSet() is used, and the different events are separated by a comma. The event combination happens when all individual events happen at the same time. It is comparable to a logic AND between the different events.

Examples

EventExtendedConfigurationTriggerSet (GPIO2.ADC2.ADCHighLimit, 3, 0, 0, 0, HEXAPOD.1.SGamma.MotionState, 0, 0, 0, 0)

This event will happen when the voltage of the GPIO.ADC2 exceeds 3 Volts during a motion of the positioner HEXAPOD.1.

EventExtendedConfigurationTriggerSet (Always, 0, 0, 0, 0, HEXAPOD.1.SGamma.MotionStart, 0, 0, 0, 0)

This event will happen at each start of a motion of the positioner HEXAPOD.1. The link with the event "always" effects that the event gets not removed anymore after the next motion has been started (see difference to first example above).

NOTE

The functions HexapodMoveAbsolute and HexapodMoveIncremental launch a synchronized motion on all six Hexapod positioners. Hence, the event EventExtendedConfigurationTriggerSet (HEXAPOD.1.SGamma.MotionStart, 0, 0, 0, 0) is also always true at the start of a HexapodMoveAbsolute() or a HexapodMoveIncremental(). The same is true for all other motion related events.

The exact meaning of the different events and event parameters is as follow:

Always:	Triggers an action ALWAYS, means each servo cycle. Event parameter 1 to 4 = 0 by default. **NOTE:** This event is PERMANENT until the next reboot. Call the EventExtendedRemove function to remove it.
Immediate:	Triggers an action IMMEDIATELY, means once during the very next servo cycle: Event parameter 1 to 4 = 0 by default. **NOTE:** This event is EPHEMERAL.
Timer:	Triggers an action every nth servo cycle, where n gets defined with the function TimerSet. Event parameter 1 to 4 = 0 by default. **NOTE:** This event is PERMANENT until the next reboot. Call the EventExtendedRemove function to remove it.
MotionStart:	Triggers an action when the motion starts. Event parameter 1 to 4 = 0 by default.

MotionEnd: Trigger an action when the motion is ended. Event parameter 1 to 4 = 0 by default. Note, MotionEnd refers to the end of the theoretic motion which is not necessarily the same as MotionDone depending on the definition (see also section 9.9).

MotionState: Triggers an action during the motion. Event parameter 1 to 4 = 0 by default.

MotionDone: Triggers an action when the Motion done is true. Event parameter 1 to 4 = 0 by default. For the exact definition of MotionDone, please refer to section 9.9. If MotionDone is set to "Theoretical", MotionDone is equal to MotionEnd.

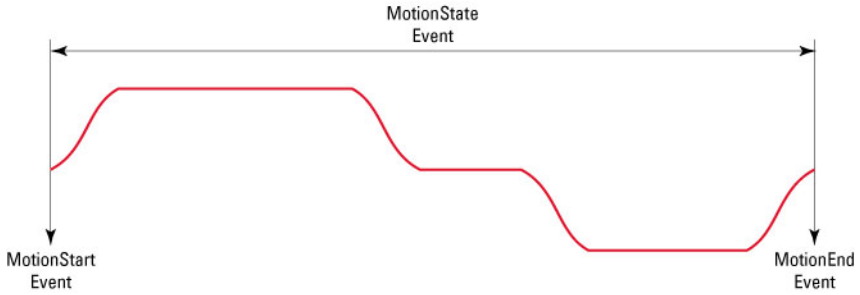


Figure 24: Motion Event.

ConstantVelocityStart: Triggers an action when the constant velocity is reached. Event parameter 1 to 4 = 0 by default.

ConstantVelocityEnd: Triggers an action when the constant velocity is finished. Event parameter 1 to 4 = 0 by default.

ConstantVelocityState: Triggers an action during the constant velocity state. Event parameter 1 to 4 = 0 by default.

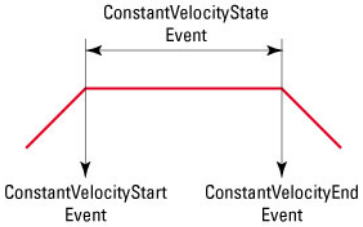


Figure 25: Constant Velocity Event.

ConstantAccelerationStart: Triggers an action when the constant acceleration is reached. Event parameter 1 to 4 = 0 by default.

ConstantAccelerationEnd: Triggers an action when the constant acceleration is finished. Event parameter 1 to 4 = 0 by default.

ConstantAccelerationState: Triggers an action during the constant acceleration state. Event parameter 1 to 4 = 0 by default.

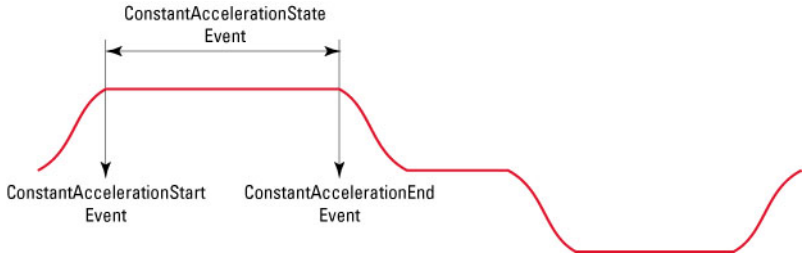


Figure 26: Constant Acceleration Event.

The same definition applies to ConstantDecelerationStart, ConstantDecelerationEnd and ConstantDecelerationState.

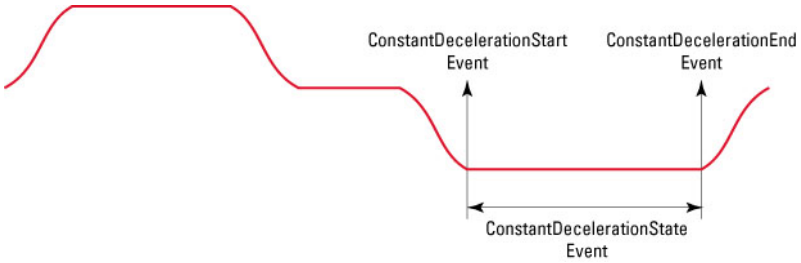


Figure 27: Constant Deceleration Event.

DILowHigh: Triggers an action when the digital input bit switches from low state to high state. The first event parameter is the bit index (0 to 15). The other event parameters are 0 by default.

DIHighLow: Triggers an action when the digital input bit switches from high state to low state. The first event parameter is the bit index (0 to 15). The other event parameters are 0 by default.

DIToggled: Triggers an action when the digital input bit switches from low to high or from high to low. The first event parameter is the bit index (0 to 15). The other event parameters are 0 by default.

ADCHighLimit: Triggers an action when the analog input value exceeds the limit. The first event parameter is the limit value in volts. The other event parameters are 0 by default.

ADCLowLimit: Triggers an action when the analog input value is below the limit. The first event parameter is the limit value in volts. The other event parameters are 0 by default.

PositionerError: Triggers an action when the current positioner error applied with the error mask results in a value different than zero. The first event parameter specifies the error mask in a decimal format. The other event parameters are 0 by default.

Code (Hexa)	Bit #	Decimal	Positioner error description
0			No error
0x00000001	0	1	General inhibition detected
0x00000002	1	2	Fatal following error detected
0x00000004	2	4	Home search time out
0x00000008	3	8	Motion done time out
0x00000010	4	16	Requested position exceed travel limits in trajectory or slave mode
0x00000020	5	32	Requested velocity exceed maximum value in trajectory or slave mode
0x00000040	6	64	Requested acceleration exceed max value in trajectory or slave mode
0x00000100	8	256	Minus end of course activated
0x00000200	9	512	Plus end of course activated
0x00000400	10	1024	Minus end of run glitch
0x00000800	11	2048	Plus end of run glitch
0x00001000	12	4096	Encoder quadrature error
0x00002000	13	8192	Encoder frequency and coherence error
0x00010000	16	65536	Hard interpolator encoder error
0x00020000	17	131072	Hard interpolator encoder quadrature error
0x00100000	20	1048576	First driver in fault
0x00200000	21	2097152	Second driver in fault

Examples

EventExtendedConfigurationTriggerSet (HEXAPOD.1.PositionerError, 2, 0, 0, 0)

This event happens when the positioner HEXAPOD.1 has a fatal following error.

EventExtendedConfigurationTriggerSet (HEXAPOD.1.PositionerError, 12, 0, 0, 0)

This event happens when the positioner HEXAPOD.1 has either a home search time out or a motion done time out.

PositionerHardwareStatus: Triggers an action when the current hardware status applied with the error mask results in a value different than zero. The first event parameter specifies the status mask in a decimal format. The other event parameters are 0 by default.

Code (Hexa)	Bit #	Decimal	Hardware status description
0x00000001	0	1	General inhibition detected
0x00000004	2	4	ZM high level
0x00000100	8	256	Minus end of run activated
0x00000200	9	512	Plus end of run activated
0x00000400	10	1024	Minus end of run glitch
0x00000800	11	2048	Plus end of run glitch
0x00001000	12	4096	Encoder quadrature error
0x00002000	13	8192	Encoder frequency or coherence error
0x00010000	16	65536	Hard interpolator encoder error
0x00020000	17	131072	Hard interpolator encoder quadrature error
0x00100000	20	1048576	First driver in fault
0x00200000	21	2097152	Second driver in fault
0x00400000	22	4194304	First driver powered on
0x00800000	23	8388608	Second driver powered on

Example

EventExtendedConfigurationTriggerSet (HEXAPOD.1.PositionerHardwareStatus, 768, 0, 0, 0)

This event happens when the positioner HEXAPOD.1 has either a plus end of run or a minus end of run detected.

10.2 Actions

There are several actions that can be triggered by the events listed above. Users have the full flexibility to trigger any action (out of the list of possible actions) at any event (out of the list of possible events). It is also possible to trigger several actions at the same event by specifying several actions, separated by a comma (,), similar to how it is done for events.

Actor		Action Name	Parameter			
Group	GPIO		1	2	3	4
	■	DOToggle	Mask			
	■	DOPulse	Mask			
	■	DOSet	Mask	Value		
	■	DACSet.CurrentPosition	Positioner name	Gain	Offset	
	■	DACSet.CurrentVelocity	Positioner name	Gain	Offset	
	■	DACSet.SetpointPosition	Positioner name	Gain	Offset	
	■	DACSet.SetpointVelocity	Positioner name	Gain	Offset	
	■	DACSet.SetpointAcceleration	Positioner name	Gain	Offset	
		ExecuteTCLScript	TCL file name	Task name	Arguments	
		KillTCLScript	Task name			
		GatheringOneData				
		GatheringRun	Nb of points	Divisor		
		GatheringRunAppend				
		GatheringStop				
		ExternalGatheringRun	Nb of points	Divisor		
■		MoveAbort				

CAUTION

Certain events like MotionState have duration. These events trigger the associated action each servo or profiler cycle as long as the event is true. For example, associating the action DOToggle with the event MotionState will toggle the value of the digital output each profiler cycle as long as the MotionState event is true.

An event doesn't reset the action after the event: For example, to set a digital output to a certain value during the constant velocity state and to set it back to its previous value afterwards, two event triggers are needed: One to set to the digital output of the desired value at the event ConstantVelocityStart and another one to set it back to its original value at the event ConstantVelocityEnd. The same effect can NOT be achieved with the sole use of the event ConstantVelocityState.

An action is entirely composed by:

[Actor].Action Name, Parameter1, Parameter2, Parameter3, Parameter4.

Not all action names have a preceding actor, but all actions have four parameters. Even if an action does not use all four parameters, all four parameters still need to be specified. The value for a not used parameter must be zero by default.

Please note, the separator between the actor and the Action Name is a dot (.). The separator between the Action Name and the parameters is a comma (,).

To define an action, use the function **EventExtendedConfigurationActionSet()**.

Example

EventExtendedConfigurationActionSet (GPIO1.DO.DOToggled, 4, 0, 0, 0)

In this case the actor is the digital output GPIO1.DO and the action is to toggle the output. The mask 4 refers to bit #3, 00000100. Hence, this action toggles the value of bit 3 on the digital output GPIO.DO.

EventExtendedConfigurationActionSet (ExecuteTCLScript, Example.tcl, 1, 0, 0)

The action ExecuteTCLScript has no preceding actor. This action will start the execution of the TCL script "Example.tcl". The task name is 1 and the TCL script has no arguments (a zero for the third parameter means no arguments).

EventExtendedConfigurationActionSet (GatheringRun, 1000, 10, 0, 0)

The action GatheringRun has no preceding actor. This action will start an internal data gathering. It will gather a total of 1000 data points, one data point every 10th servo cycle, means one data point every 10/CorrectorISRPeriod.

It is also possible to trigger several actions at the same event. To do so, just define another action in the SAME function. Several actions are separated by a comma (,).

Example

EventExtendedConfigurationTriggerSet (HEXAPOD.1.PositionerError, 2, 0, 0, 0)

EventExtendedConfigurationActionSet (ExecuteTCLScript, ShutDown.tcl, 1, 0, 0, ExecuteTCLScript, ErrorDiagnostic.tcl, 2, 0, 0)

EventExtendedStart ()

In this example the TCL scripts ShutDown.tcl and ErrorDiagnostic.tcl are executed when a fatal following error is detected on the positioner HEXAPOD.1.

The exact meaning of the different actions and action parameters is as follow:

DOToggle: This action is used to reverse the value of one or many bits on the Digital Output. When using this action with an event that has some duration (for example motion state) the value of the bits will be toggled each profiler cycle as long as the event occurs.

Action Parameter #1 – Mask	The mask defines which bits on the GPIO output will be toggled (change their value). For example, if the GPIO output is an 8 bit output and the mask is set to 4 then the equivalent binary number is 00000100. So as an action, the bit #3 will be toggled.
Action Parameter #2 to #4	These parameters must be 0 by default.

DOPulse: This action is used to generate a positive pulse on the Digital Output. The duration of the pulse is 1 microsecond. To function, the bits on which the pulse is generated should be set to zero before. When using this action with an event that has some duration (for example motion state) a 1 µs pulse will be generated each cycle of the Motion Profiler as long as the event occurs.

Action Parameter #1 – Mask	The mask defines on which bits on the GPIO output the pulse will be generated. For example, if the GPIO output is an 8 bit output and the mask is set to 6 then the equivalent binary number is 00000110. So as an action, a 1 µs pulse will be generated on bit #2 and #3 of the GPIO output.

Action Parameter #2 to #4	These parameters must be 0 by default.

DOSet: This action is used to modify the value of bit(s) on a Digital Output.

Action Parameter #1 – Mask	The mask defines which bits on the GPIO output are being addressed. For example, if the GPIO output is an 8 bit output and the mask is set to 26 then the equivalent binary number is 00011010. Therefore with a Mask setting of 26, only the bits # 2, #4 and #5 are being addressed on the GPIO output.
Action Parameter #2 – Value	This parameter sets the value of the bits that are being addressed according to the Mask setting. So for example since a Mask setting of 26, bits #2, #4 and #5 can be modified, a value of 8 (00001000) will set the bits #2 and #5 to 0 and the bit #4 to 1.
Action parameter #3 and #4	These parameters must be 0 by default.

DACSet.CurrentPosition and **DACSet.SetpointPosition:** This action sets a voltage on the Analog output in relation to the actual (current) or theoretical (Setpoint) position. The gain and the offset are used to calibrate the output. This action makes most sense with events that have duration (always, MotionState, etc.) as the analog output will be updated each servo cycle or each profiler cycle as long as the event lasts. When used with events that have no duration (like MotionStart or MotionEnd), the analog output gets only updated once and this value is hold until the next change.

Action Parameter #1 – Positioner Name	This parameter defines the name of the positioner which position value is used.
Action Parameter #2 – Gain	The position value is multiplied by the gain value. For example, if the gain is set to 10 and the position value is 1 mm, then the output voltage is 10 V.
Action Parameter #3 – Offset	The offset value is used to correct for any voltage that may be present on the Analog output.

Analog output = Position value * gain + offset

Action parameter #4	This parameter must be 0 by default.

DACSet.CurrentVelocity and **DACSet.SetpointVelocity:** This action sets a voltage on the Analog output in relation to the actual (current) or theoretical (Setpoint) velocity. The gain and the offset are used to calibrate the output. This action makes most sense with events that have some duration (Always, MotionState, etc.) as the analog output will be updated each servo cycle or each profiler cycle as long as the event lasts. When used with events that have no duration (like MotionStart or MotionEnd), the analog output gets only updated once and this value is hold until its next change.

Action Parameter #1 – Positioner Name	This parameter defines the name of the positioner which Velocity value is used.
Action Parameter #2 – Gain	The Velocity value is multiplied by the gain value. For example if the gain is set to 10 and the velocity value is 1 mm/s, then the output voltage is 10 V.
Action Parameter #3 – Offset	The offset value is used to correct for any voltage that may be present on the Analog output.

Analog output = Velocity value * gain + offset

Action parameter #4 — This parameter must be 0 by default.

DACSet.SetpointAcceleration: This action is used to output a voltage on the Analog output to form an image of the theoretical acceleration. The gain and the offset are used to calibrate this image. This action makes most sense with events that have some duration (Always, MotionState, etc.) as the analog output will be updated each servo cycle or each profiler cycle as long as the event lasts. When used with events that have no duration (like MotionStart or MotionEnd), the analog output gets only updated once and holds this value until its next change.

Action Parameter #1 – Positioner Name — This parameter defines the name of the positioner which SetpointAcceleration is used to output on the analog output.

Action Parameter #2 – Gain — The SetpointAcceleration is multiplied by the gain value. For example if the gain is set to 10 and the corrected SetpointAcceleration is 1 mm/s2 then the output voltage will be 10 V.

Action Parameter #3 – Offset — The offset value is used to correct for any voltage that may be present on the Analog output.

Analog output = SetpointAcceleration value * gain + offset

Action parameter #4 — This parameter must be 0 by default.

NOTE

The gain can be any constant value used to scale the output voltage and the offset value can be any constant value used to correct for any offset voltage on the analog output.

ExecuteTCLScript: This action executes a TCL script on an event.

Action Parameter #1 – TCL File Name — This parameter defines the file name of the TCL program.

Action Parameter #2 – TCL Task Name — Since several different or even the same TCL scripts can run simultaneously, the TCL Task Name is used to track individual TCL programs. For example, the TCL Task Name allows stopping a particular program without stopping all other TCL programs that run simultaneously.

Action Parameter #3 – TCL Arguments List — The Argument list is used to run the TCL scripts with input parameters. For the argument parameter, any input can be given (number, string). These parameters are used inside the script. To get the number of arguments use $tcl_argc" inside the script. To get each argument use "$tcl_argc($i)" inside the script. For example, this parameter can be used to specify a number of loops inside the TCL script. A zero (0) for this parameter means there are no input arguments.

Action parameter #4 — This parameter must be 0 by default.

KillTCLScript: This action stops a TCL script on an event.

Action parameter #1 – Task name	This parameter defines which TCL script is stopped. Since several different or even the same TCL scripts can run simultaneously, the TCL Task Name is used to track individual TCL programs.
Action parameter #2 to #4	These parameters must be 0 by default.

GatheringOneData: This action acquires one data as defined by the function GatheringConfigurationSet. Different than the GatheringRun (see next action), which generates a new gathering file, the GatheringOneData appends the data to the current gathering file stored in memory. In order to store the data in a new file, you first need to launch the function GatheringReset, which deletes the current gathering file from memory.

Action parameter #1 to #4	These parameters must be 0 by default.

GatheringRun: This action starts an internal gathering. It requires that an internal gathering was previously configured with the function GatheringConfigurationSet. The gathering must be launched by a punctual event and does not work with events that have duration.

Action Parameter #1 – NbPoints	This parameter defines the number of data acquisitions. NbPoints multiplied with the number of gathered data types must be smaller than 1,000,000. For instance, if 4 types of data are collected, NbPoints can not be larger than 250,000 (4*250,000 = 1,000,000).
Action Parameter #2 – Divisor	This parameter defines the frequency for the gathering in relation to the servo frequency of the system (adjustable up to 10 kHz). This parameter has to be an integer and greater or equal to 1. For instance, with a servo frequency of 10 kHz, if the parameter is set to 10, then the gathering will take place every 10^{th} servo cycle or at a rate of 1 kHz (10 kHz/10) or at every 1 msec.
Action Parameter #3 and #4	These parameters must be 0 by default.

GatheringRunAppend: This action continues a gathering previously stopped with the action GatheringStop, see next action.

Action parameter #1 to #4	These parameters must be 0 by default.

GatheringStop: This action halts a data gathering previously launched by the action GatheringStart. Use the action GatheringRunAppend to continue the data gathering. Please note, that the action GatheringStop does not automatically store the gathered data from the buffer to the flash disk of the controller. For doing so, please use the function GatheringStopAndSave. For more details about data gathering, please refer to chapter 13.0: “Data Gathering”.

Action parameter #1 to #4	These parameters must be 0 by default.

ExternalGatheringRun: This action starts an external gathering. It requires that an external gathering was previously configured with the function GatheringExternalConfigurationSet. The gathering must be launched by a punctual event and does not work with events that have duration.

Action Parameter #1 – NbPoints	This parameter defines the number of data acquisitions. NbPoints multiplied with the number of gathered data types must be smaller than 1,000,000. For instance, if 4 types of data are collected, NbPoints can not be larger than 250,000 (4*250,000 = 1,000,000).
Action Parameter #2 – Divisor	This parameter defines every Nth number of trigger input signal at which the gathering will take place. This parameter has to be an integer and greater or equal to 1. For example if the divisor is set to 5 then gathering will take place every 5th trigger on the trigger input signal.
Action Parameter #3 and #4	These parameters must be 0 by default.

NOTE

For further details on data gathering, see chapter 13.0: "Data Gathering".

MoveAbort: This action allows to stop (abort) a motion on an event. It is similar to sending a MoveAbort() function on the event. After breaking, the group is in the READY state.

Action Parameter #1 to #4	These parameters must be 0 by default.

10.3 Functions

The following functions are related to the event triggers:

- **EventExtendedConfigurationTriggerSet ()**: This function configures one or several events. In case of several events, the different events are separated by a comma (,) in the argument list. Before activating an event, one or several actions must be configured with the function EventExtendedConfigurationActionSet(). Only then, the event and the associated action(s) can get activated with the function EventExtendedStart().
- **EventExtendedConfigurationTriggerGet ()**: This function returns the event configuration defined by the last EventExtendedConfigurationTriggerSet() function.
- **EventExtendedConfigurationActionSet ()**: This function associates an action to the event defined by the last EventExtendedConfigurationTriggerSet() function.
- **EventExtendedConfigurationActionGet ()**: This function returns the action configuration defined by the last EventExtendedConfigurationActionSet() function.
- **EventExtendedStart ()**: This function launches (activates) the last configured event and the associated action(s) defined by the last EventExtendedConfigurationTriggerSet() and EventExtendedConfigurationActionSet() and returns an event identifier. When activated, the HXP controller checks for the event each servo cycle (or each profiler cycle for those events that are motion related) and triggers the action when the event occurs. Hence, the latency between the event and the action is equal to the servo cycle time as short as 100 µs or equal to the profiler cycle time as short as 400 µs for motion related events. For events with duration, it means also that the same action is triggered each servo cycle or each profiler cycle, as long as the event is happening.

  Event triggers (and their associated action) get automatically removed after the event configuration has happened at least once and is no longer true anymore. The

only exception is if the event configuration contains any of the permanent events “Always” or “Trigger”. In that case the event trigger will always stay active. With the function EventExtendedRemove(), any event trigger can get removed.

- **EventExtendedWait ()**: This function halts a process (essentially by blocking the socket) until the event defined by the last EventExtendedConfigurationTriggerSet() occurs.
- **EventExtendedRemove ()**: This function removes the event trigger associated to the defined event identifier.
- **EventExtendedGet ()**: This function returns the event configuration and the action configuration associated to the defined event identifier.
- **EventExtendedAllGet ()**: This function returns for all active event triggers the event identifier, the event configuration and the action configuration. The details of the different event triggers are separated by a comma (,).

10.4 Examples

Below is a table that shows possible events that can be associated with possible actions. Some of these examples however, may have unwanted results. Since the HXP controller provides great flexibility to trigger almost any action at any event, the user must be aware of the possible unwanted effects.

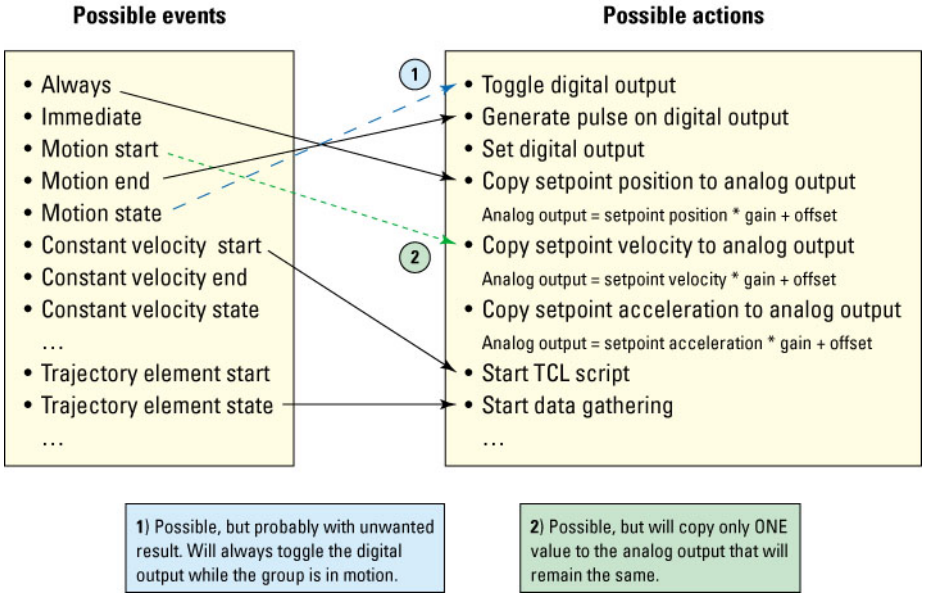


Figure 28: Possible Events.

Examples

1. **EventExtendedConfigurationTriggerSet (HEXAPOD.1.SGamma.ConstantVelocityStart, 0, 0, 0, 0)**

   EventExtendedConfigurationActionSet (GPIO1.DO.DOSet, 4, 4, 0, 0)

   EventExtendedStart()

   HexapodMoveAbsolute (HEXAPOD, Work, 10, 20, 5, 0, 0, 0)

   In this example, when positioner HEXAPOD.1 reaches constant velocity, bit #3 on the digital output on connector number 1 is set to 1 (Note: 4 = 00000100). Note, that the state of the bit will not change when the constant velocity of the positioner is ended. In order to do so, a second event trigger would be required (see next example). Note also that the function HexapodMoveAbsolute launches a synchronized motion on all Hexapod struts. Hence, when positioner HEXAPOD.1

has reached its constant velocity, all other positioners have reached their constant velocity as well.

2. **EventExtendedConfigurationTriggerSet (HEXAPOD.1.SGamma.ConstantVelocityStart, 0, 0, 0, 0)**

   EventExtendedConfigurationActionSet (GPIO1.DO.DOSet, 4, 4, 0, 0)

   EventExtendedStart()

   EventExtendedConfigurationTriggerSet (HEXAPOD.1.SGamma.ConstantVelocityEnd, 0, 0, 0, 0)

   EventExtendedConfigurationActionSet (GPIO1.DO.DOSet, 4, 0, 0, 0)

   EventExtendedStart()

   HexapodMoveAbsolute (HEXAPOD, Work, 10, 20, 5, 0, 0, 0)

   In this example, when positioner HEXAPOD.1 reaches constant velocity, bit #3 on the digital output on connector number 1 is set to 1 (Note: 4 = 00000100) and when the constant velocity of the positioner HEXAPOD.1 is over, bit #3 will be set to zero. Note, that the same effect can not be reached with the event name ConstantVelocityState.
   When both events have happened, the event triggers will get automatically removed. In order to trigger the same action at each motion, it is required to link the events with the event “Always” (see next example). This link will avoid that the event trigger gets removed after it is not happening anymore.

3. **EventExtendedConfigurationTriggerSet (Always, 0, 0, 0, 0, HEXAPOD.1.SGamma.ConstantVelocityStart, 0, 0, 0, 0)**

   EventExtendedConfigurationActionSet (GPIO1.DO.DOSet, 4, 4, 0, 0)

   EventExtendedStart()

   EventExtendedConfigurationTriggerSet (Always, 0, 0, 0, 0, HEXAPOD.1.SGamma.ConstantVelocityEnd, 0, 0, 0, 0)

   EventExtendedConfigurationActionSet (GPIO1.DO.DOSet, 4, 0, 0, 0)

   EventExtendedStart()

   HexapodMoveAbsolute (HEXAPOD, Work, 10, 20, 5, 0, 0, 0)

   HexapodMoveAbsolute (HEXAPOD, Work, 0, 0, 0, 0, 0, 0)

   In this example, when positioner HEXAPOD.1 reaches constant velocity, bit #3 on the digital output on connector number 1 is set to 1 (Note: 4 = 00000100) and when the constant velocity of the positioner HEXAPOD.1 is over, bit #3 will be set to zero. Different than in the previous example, here the concatenate with the event “Always” avoids that the event trigger gets removed after the event is over. Hence, the state of the bit #3 will change with every beginning and with every end of the constant velocity state of a motion.

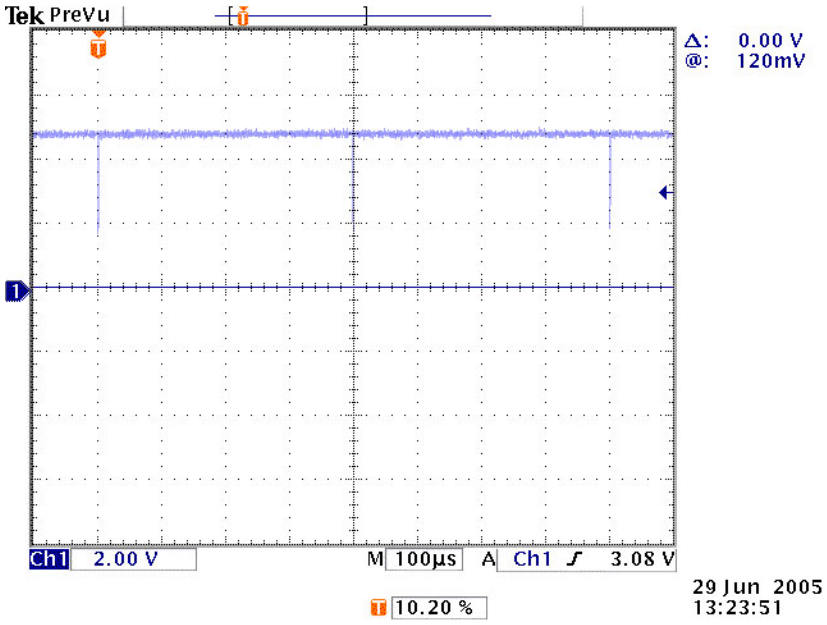
4. **EventExtendedConfigurationTriggerSet (HEXAPOD.1.SGamma.ConstantVelocityState, 0, 0, 0, 0)**

   EventExtendedConfigurationActionSet (GPIO1.DO.DOPulse, 255, 0, 0, 0)

   EventExtendedStart()

   HexapodMoveAbsolute (HEXAPOD, Work, 10, 20, 5, 0, 0, 0)

   In this example, during the constant velocity state of the positioner HEXAPOD.1, 1 µs pulses are generated on all 8 bits on the digital output on connector number 1 every cycle of the motion profiler (Note: 255 = 11111111). Suppose the cycle time of the motion profiler is 400 µs, so pulses are generated every 400 µs (see picture below).



5. **EventExtendedConfigurationTriggerSet (Always, 0, 0, 0, 0)**

   EventExtendedConfigurationActionSet (GPIO2.DAC1.DACSet.SetpointPosition, HEXAPOD.1, 0.1, -10, 0)

   GPIO2.DAC2.DACSet.SetpointVelocity, HEXAPOD.1, 0.5, 0, 0)

   EventExtendedStart()

   In this example, the analog output #1 on GPIO2 will always output a voltage in relation to the SetpointPosition of the positioner HEXAPOD.1, and the output #2 on GPIO2 will always output a voltage in relation to the SetpointVelocity of the same positioner. The gain on output #1 is set to 0.1 V/unit and the offset to -10 V. This means when the stage is at the position 0 unit, a voltage of -10 V will be output. When the stage is at the position 10 units, a voltage of -9V will be outputted. Here, the event “Always” makes that these values will get updated every servo cycle. If instead of the event “Always” the event “Immediate” will be used, only the most recent values will be output and kept. If instead of the event “Always” a motion related event such as MotionState will be used, the update will only happen every profiler cycle, rather than the faster servo rate.

6. **TimerSet(Timer1,10000)**

   EventExtendedConfigurationTriggerSet (Timer1.Timer, 0, 0, 0, 0)

   EventExtendedConfigurationActionSet (GPIO1.DO.DOToggle, 255, 0, 0, 0)

   EventExtendedStart()

   EventExtendedRemove(1)

   The function Timer() sets the Timer1 at every 10 000th servo cycle, or at one second with a servo rate of 10 kHz. Hence, in this example, every second all bits on digital output on connector number 1 will be toggled (Note: 255 = 11111111). The event Timer is permanent. In order to remove the event trigger, use the function EventExtendedRemove() with the ssociated event identifier (1 in this case).

7. **GatheringConfigurationSet(HEXAPOD.1.CurrentPosition)**

   EventExtendedConfigurationTriggerSet (HEXAPOD.1.SGamma.MotionStart,0,0,0,0)

   EventExtendedConfigurationActionSet(GatheringRun,20,1000,0,0)

   EventExtendedStart()

   HexapodMoveAbsolute (HEXAPOD, Work, 10, 20, 5, 0, 0, 0)

   GatheringStopAndSave()

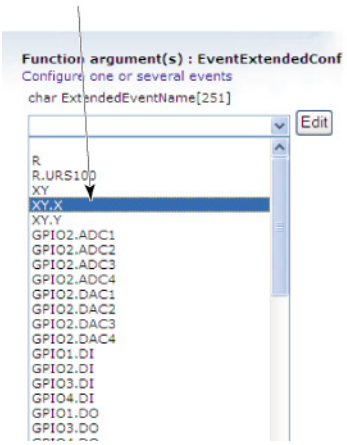
   In this example, an internal data gathering of 20 data points every 0.1 second (every 1000th servo cycle with the servo rate at 10 kHz) is launched with the start of the next motion of the positioner HEXAPOD.1. The type of data that gets gathered is defined with the function GatheringConfigurationSet (CurrentPosition of positioner HEXAPOD.1). To store the data from internal memory to the flash disk in the HXP controller, you need to send the function GatheringStopAndSave(). The GatheringRun deletes the current data file in the internal memory (in contrast to the GatheringOneData which appends data to the current file). Also, the function GatheringStopAndSave() stores the data file under a default name Gathering.dat on the flash disk of the HXP controller and will overwrite any older file of the same name in the same folder. Hence, make sure that you store your valuable data files under different name after the GatheringStopAndSave().

NOTE

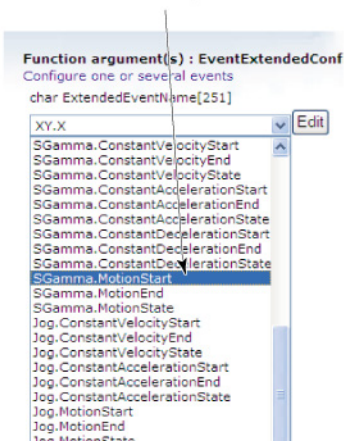
When using the function EventExtendedConfigurationTriggerSet() or EventExtendedConfigurationActionSet () from the terminal screen of the HXP utility, the syntax for one parameter is not directly accessible. For instance, for the event HEXAPOD.1.SGamma.MotionStart, first select HEXAPOD.1 from the choice list. Then, click on the choice field again and select SGammaMotionStart. See also following screen shots.

For specifying more than one data type, use the ADD button. Select the next parameter as described above.

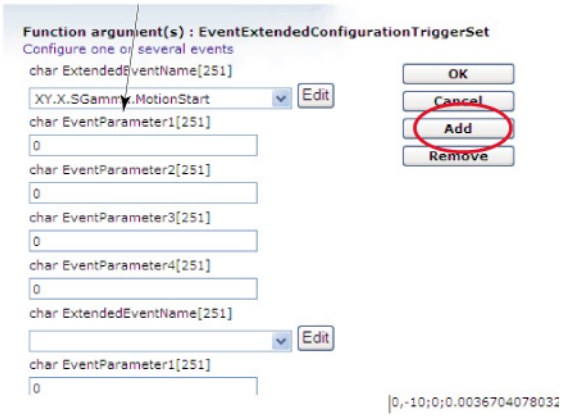
Step 1:
Select the positioner name and click.



Step 2:
Click in the choice field again and select the parameter name.



Step 3:
Define event parameters.
To add another event, click ADD, else click OK.



11.0 Data Gathering

The HXP controller provides four methods for data gathering:

1. Time based (internal) data gathering. With this method one data set is gathered every n^{th} servo cycle.
2. Event based (internal) data gathering. With this method one data set is gathered on an event.
3. Function based (internal) data gathering. With this method one data set is gathered by a function.
4. Trigger based (external) data gathering. With this method one data set is gathered every n^{th} external trigger input.

With method 1, 2, and 3 we are also referring to an internal or servo cycle synchronous data gathering. With the trigger based data gathering we are also referring to an external data gathering as the event that triggers the data gathering, the receive of a trigger input, is asynchronous to the servo cycle.

The servo rate of the HXP controller can be found in the system.ref file stored in the controller. To access the system.ref file, the user must establish a FTP contection to the controller. Refer to chapter 5.0 FTP (File Transfer Protocol) Connection (Bernard could you make this a hyperlink in PDF file) for detailed instructions. Once, you have logged in as Adminstrator, open the Config folder which contains the system.ref file.

Example System.ref

```
[GENERAL]
FirmwareName = MainController
ExternalModuleNames =
CorrectorISRPeriod = 175e-6 ; Servo rate in seconds
IRQDelay = 6e-6 ; seconds
DACUpdateDelay = 154e-6 ; seconds
ProfileGeneratorISRRatio = 14 ;
ServitudesISRRatio = 10
GatheringBufferSize = 1000000
DelayBeforeStartup = 0 ; seconds
SingleAxisGroupOption = Enabled ; Enabled or Disabled - Enabled is only allowed on 8 axes system (HXP + 2)
```

The time based, the event based and the function based data gathering store the data in common file called gathering.dat. The trigger based (external) data gathering stores the data in different file, called ExternalGathering.dat. The type of data that can be gathered differs also between the internal and the external data gathering.

Before starting any data gathering the type of data to be gathered needs to be defined using the functions **GatheringConfigurationSet()** (in case of an internal data gathering) or **GatheringExternalConfigurationSet()** (in case of an external data gathering).

During the data gathering new data is appended to a buffer. With the functions **GatheringCurrentNumberGet()** and **GatheringExternalCurrentNumberGet()** the current number of data sets in this buffer and the maximum possible number of data sets that fits into this buffer can be recalled. The maximum possible number of data sets is equal to 1 000 000 divided by the number of data types belonging to one data set.

The function **GatheringDataGet(index)** returns one set of data from the buffer. Here, the index 0 refers to the 1st data set, the index (n-1) to the n-th data set. When using this

function from the Terminal screen of the HXP utilities, the different data types belonging to one data line are separated by a semicolon (;)

To save the data from the buffer to the flash disk of the HXP controller, use the functions **GatheringStopAndSave()** and **GatheringExternalStopAndSave()**. These functions will store the gathering files in the..\Admin\Public folder of the HXP controller under the name Gathering.dat (with function GatheringStopAndSave() for an internal gathering) or GatheringExternal.dat (with function GatheringExternalStopAndSave() for an external gathering).

CAUTION

The functions GatheringStopAndSave() and GatheringExternalStopAndSave() overwrite any older files with the same name in the ..\Admin\Public folder. After a data gathering, it is required to rename or better, to relocate valid data files using an ftp link to the HXP controller (see also chapter 5.0: "FTP (File Transfer Protocol) Connection").

A gathering file can have a maximum of 1,000,000 data entries and a maximum of 25 different data types. The first line of the data file contains the sample period in seconds (minimum period = CorrectorISRPeriod), the second line contains the names of the data type(s) and the other lines contain the acquired data. A prototype of a sample file is shown below.

Gathering.dat

```
SamplePeriod          0                    0
GatheringTypeA  GatheringTypeB  GatheringTypeC
ValueA1               ValueB1              ValueC1
ValueA2               ValueB2              ValueC2
…                     …                    …
ValueAN               ValueBN              ValueCN
```

11.1 Time Based (Internal) Data Gathering

The data for the time based gathering get latched by an internal interrupt related to the servo cycle of the system (adjustable up to 10 kHz). The function GatheringConfigurationSet() defines which type of data will be stored in the data file. The following table lists all data type(s) that can be collected:

Actor			Parameter
PositionerName	**CoordinateName**	**GPIO**	
■	■		SetpointPosition
■	■		CurrentPosition
■			FollowingError
■			SetpointVeolicty
■			CurrentVelocity
■			SetpointAcceleration
■			CurrentAcceleration
■			CorrectorOutput
		■	DI
		■	DO
		■	ADC
		■	DAC

A data is defined by **Actor.Parameter**.

The PositionerName refers to the names of the positioners defined in the system.ini (e.g. HEXAPOD.1, HEAXPOD.2, etc.). These are the struts of the Hexapod. The data HEXAPOD.1.CurrentPosition, for instance refers to the length of the HEXAPOD.1 strut.

The CoordinateName refers to the X, Y, Z, U, V and W coordinates of the Hexapod group (position of Tool in Work, see chapter 8.0 for details). The coordinate names are:

GroupName.X
GroupName.Y
GroupName.Z
GroupName.U
GroupName.V
GroupName.W

In the default configuration of the HXP, the GroupName is HEXAPOD, hence HEXAPOD.X, HEXAPOD.Y, etc.

GPIO refers to the different GPIO plugs of the HXP (GPIO1, GPIO2, GPIO3, and GPIO4). Please note that not all GPIO plugs have digital inputs (DI), digital outputs (DO), analog inputs (ADC) and analog outputs (DAC). For a complete list of available I/O's accessible with the HXP, please refer to chapter 18.0.

It is possible to start the gathering either by function call or at an event. The following sequence of functions is used for a time based data gathering started by function call:

GatheringConfigurationSet()

GatheringRun()

The following sequence of functions is used to start a time based data gathering at an event:

GatheringConfigurationSet()

EventExtendedConfigurationTriggerSet()

EventExtendedConfigurationActionSet()

EventExtendedStart()

A function which triggers the action, for instance a HexapodMoveAbsolute ().

When all data is gathered, use the function **Gathering StopAndSave()** to save the data from the buffer to the flash disk of the HXP controller.

Other functions associated with internal Gathering are:

GatheringConfigurationGet()

GatheringCurrentNumberGet()

GatheringDataGet()

GatheringDataMultipleLinesGet()

GatheringStop()

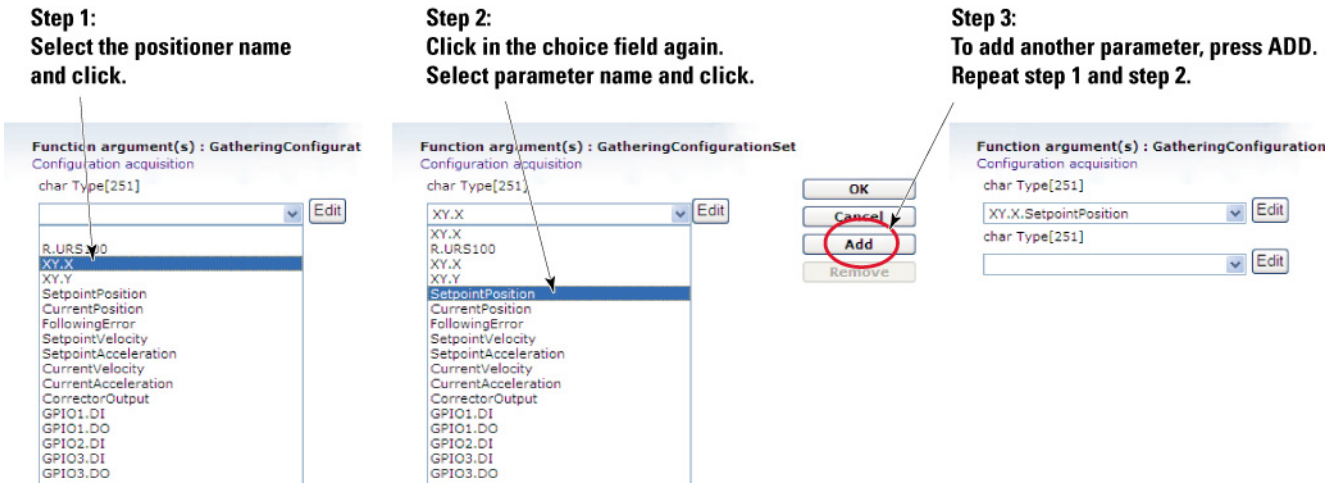
GatheringRunAppend()

See Programmer's Manual for details on functions.

NOTE

When using the function GatheringConfigurationSet() from the terminal screen of the HXP utility, the syntax for one parameter is not directly accessible. For instance, for the parameter HEXAPOD.1.SetpointPosition, first select HEXAPOD.1 from the choice list. Then, click on the choice field again and select SetpointPosition. See also screen shots on the next page.

For specifying more than one data type, use the ADD button. Select the next parameter as described above.



Example 1

NOTE

In this example, the servo rate is defined at 10 kHz.

Using the terminal screen of the HXP utility, this example shows the sequence of functions to accomplish a time based data gathering triggered at an event.

GroupInitialize(HEXAPOD)

GroupHomeSearch(HEXAPOD)

GatheringConfigurationSet(HEXAPOD.X.SetpointPosition, HEXAPOD.X.CurrentPosition)

The 2 data HEXAPOD.X.SetpointPosition and HEXAPOD.X.CurrentPosition will be gathered.

EventExtendedConfigurationTriggerSet (HEXAPOD.X.SGamma.MotionStart,0,0,0,0)

EventExtendedConfigurationActionSet(GatheringRun,5000,10,0,0)

EventExtendedStart()

HexapodMoveAbsolute (HEXAPOD, Work, 10, 0, 0, 0, 0, 0)

GatheringStopAndSave()

In this example, a gathering is started when the positioner HEXAPOD.X starts its next motion, for instance by the functions HexapodMoveAbsolute () (that starts a synchronized motion on all Hexapod positioners). The types of data being collected are the SetpointPosition and the CurrentPosition of the X coordinate of the Hexapod group HEXAPOD. A total of 5000 data sets are collected, one data point every 10th servo cycles, or one data point every 10/10000 s = 0.001 s.

Example 2

NOTE

In this example, the servo rate is defined at 10 kHz.

Using the terminal screen of the HXP utility, this example shows the sequence of functions to accomplish a time based data gathering started by function call.

GroupInitialize(HEXAPOD)

GroupHomeSearch(HEXAPOD)

GatheringConfigurationSet(HEXAPOD.X.SetpointPosition, HEXAPOD.X.CurrentPosition)

GatheringRun (5000,10)

HexapodMoveAbsolute (HEXAPOD, Work, 10, 0, 0, 0, 0, 0)

GatheringStop ()

GatheringStopAndSave ()

In this example, the gathering is started by function call. The SetpointPosition and the CurrentPosition of the X coordinate of the Hexapod group HEXAPOD are gathered at a rate of 1 kHz (every 10th servo cycle, 10 kHz servo cycle rate). The data gathering is stopped after the HexapodMoveAbsolute is completed.

The gathering will stop automatically once the number of points specified has been collected. However, data will not be saved automatically to a file. The function **GatheringStopAndSave()** has be to used to save the data to a file.

The function **GatheringRun()** starts always a new internal data gathering and deletes any previous internal gathering data hold in the buffer. If you want to append data to the file use the function **GatheringRunAppend()** instead.

It is also possible to halt a data gathering at an event. To do so, define another event trigger and assign the action GatheringStop to that event. Use another event trigger and assign the action GatheringRunAppend to continue with the gathering. For details, see chapter 12.0: “Event Triggers“.

11.2 Event Based (Internal) Data Gathering

The event based gathering provides a method to gather data at an event. For instance, gathering data at a certain value of a digital or analog input, during a constant velocity state of a motion or on a trajectory pulse.

The event based data gathering uses the same file as the time based and the function based data gathering (see sections 13.1 and 13.3). However, unlike the time based gathering, the event based gathering appends data to the existing file in memory. This allows gathering data during several periods or even with different methods in one common file, see examples. To start data gathering in a new file, use the function **GatheringReset()**, which deletes the current gathering file from memory.

The data type(s) are the same as for time based data gathering, see chapter 13.1 for details.

The following sequence of functions is used for an event based data gathering:

GatheringReset()

GatheringConfigurationSet()

EventExtendedConfigurationTriggerSet()

EventExtendedConfigurationActionSet(GatheringOneData,0,0,0,0)

EventExtendedStart()

…

Use the function GatheringStopAndSave() to store the gathered file from the buffer to the flash disk of the HXP controller.

Other functions associated with the event based gathering are:

GatheringConfigurationGet()

GatheringCurrentNumberGet()

GatheringDataGet()

Please refer to the programmer's manual for details.

Example 1

NOTE

In this example, the servo rate is defined at 10 kHz.

GatheringReset()

Deletes gathering buffer.

GatheringConfigurationSet(HEXAPOD.X.SetpointPosition, HEXAPOD.X.CurrentPosition, GPIO2.ADC1)

The 3 data HEXAPOD.X.SetpointPosition, HEXAPOD.X.CurrentPosition and GPIO2.ADC1 will be gathered.

EventExtendedConfigurationTriggerSet(GPIO2.ADC1.ADCHighLimit, 5,0,0,0)

EventExtendedConfigurationActionSet(GatheringOneData,0,0,0,0)

EventExtendedStart()

The data gathering starts when the value of the GPIO2.ADC1 exceeds 5 Volts. One set of data will be gathered each servo cycle or every 100 µs (as the event is checked each servo cycle). The data gathering automatically stops when the value of the GPIO2.ADC1 falls below 5V again, as the event is automatically removed then (see chapter 12.0: "Event Triggers" for details).

Example 2

NOTE

In this example, the servo rate is defined at 10 kHz.

TimerSet(Timer1, 10)

Sets the timer 1 to 10 servo ticks, means every 1 ms.

GatheringReset()

Deletes gathering buffer.

GatheringConfigurationSet(HEXAPOD.X.SetpointPosition, HEXAPOD.X.CurrentPosition, GPIO2.ADC1)

The 3 data HEXAPOD.X.SetpointPosition, HEXAPOD.X.CurrentPosition and GPIO2.ADC1 will be gathered.

EventExtendedConfigurationTriggerSet(Timer1,0,0,0,0, GPIO2.ADC1.ADCHighLimit,5,0,0,0)

EventExtendedConfigurationActionSet(GatheringOneData,0,0,0,0)

EventExtendedStart()

Different than the previous example, here the event ADCHighLimit is linked to the event Timer1. This has two effects. First, the event gets permanent as the event timer is permanent. Second, one set of data is gathered only every 10 ms (combination of events

must be true). For details on the event definition, please see chapter 12.0: “Event Triggers“.

As a result, in this example, one set of data is gathered every 10 ms whenever the value of the GPIO2.ADC1 exceeds 5 Volts.

11.3 Function-Based (Internal) Data Gathering

The function based gathering provides a method to gather one set of data by a function. It uses the same file as the time based and the Event based data gathering, see chapters 13.1, 13.2 and 13.3 for details. At receipt of the function, one set of data is appended to the gathering file in memory.

The data type(s) that can be collected with the event based gathering are the same as for the time based and the event based gathering, see chapter 13.1 for details.

Example

GatheringReset()

Deletes gathering buffer.

GatheringConfigurationSet(HEXAPOD.X.SetpointPosition, HEXAPOD.X.CurrentPosition)

The 2 data HEXAPOD.X.SetpointPosition and HEXAPOD.X.CurrentPosition will be gathered.

GatheringDataAcquire()

Gathers one set of data.

GatheringCurrentNumberGet()

This function will return 1, 500000; 1 set of data acquired, max. 500 000 sets of data can be acquired.

GatheringDataAcquire()

GatheringDataAcquire()

GatheringCurrentNumberGet()

This function will return 3, 500000; 3 sets of data acquired, max. 500 000 sets of data can be acquired.

11.4 Trigger Based (External) Data Gathering

The trigger based data gathering allows acquiring position and analog input data at receive of an external trigger input (TRIG IN connector at the HXP, see section 22.0 for more details).

The position data is latched by dedicated hardware. The jitter between the trigger signal and the acquisition of the position data is less than 50 ns. The analog inputs, however, are only latched by an internal interrupt at the servo rate (adjustable up to 10 kHz) and the HXP will store the most recent value. Hence, the acquired analog input data might be as old as one period of the servo cycle.

NOTE

There must be a minimum time of one period of the servo rate between two successive trigger inputs.

The data of the trigger based (external) data gathering is stored in a file named ExternalGathering.dat, which is different from the file used for the internal data gathering (Gathering.dat). Hence, internal and external data gathering can be used at the same time.

The function **GatheringExternalConfigurationSet()** defines which type of data will be gathered and stored in the data file. The following data types that can be collected:

PositionerName.ExternalLatchPosition.
GPIO2.ADC1
GPIO2.ADC2
GPIO2.ADC3
GPIO2.ADC4

The ExternalLatchPosition refers to the uncorrected encoder position of the Hexapod struts, means no error corrections are taken into account. For devices with RS422 differential encoders, the resolution of the position information is equal to the encoder resolution.

For devices with sine/cosine 1Vpp analog encoder interface, the resolution is equal to the encoder scale pitch divided by the value of the positioner hard interpolator, see function **PositionerHardInterpolatorFactorGet()**. Its value is set to 20 by default; the maximum allowed value is 200. Please refer to the Programmer's Manual for details.

The external latch positions require that the device has an encoder. No position data can be latched with this method for devices that have no encoder.

GPIO2.ADC1 to GPIO2.ADC4 refer to the 4 analog input channels on GPIO2.

The following sequence of functions is used for a trigger based data gathering:

GatheringExternalConfigurationSet()

EventExtendedConfigurationTriggerSet()

EventExtendedConfigurationActionSet()

EventExtendedStart()

Other functions associated with the event based gathering are:

GatheringConfigurationGet()

GatheringCurrentNumberGet()

GatheringExternalDataGet()

Please refer to the Programmer's Manual for details.

Example

GatheringExternalConfigurationSet(HEXAPOD.1.ExternalLatchPositio n, GPIO2.ADC1)

EventExtendedConfigurationTriggerSet(Immediate,0,0,0,0)

EventExtendedConfigurationActionSet(ExternalGatheringRun,100,2,0,0)

EventExtendedStart()

In this example, a trigger based (external) gathering is started immediately (with the function EventExtendedStart()). The types of data being collected are the encoder position of the Hexapod strut HEXAPOD.1 and the value of the GPIO2.ADC1. A total of 100 data sets are collected; one set of data at each second trigger input. The gathering will stop automatically after the 100th data acquisition. Use the function **GatheringExternalStopAndSave()** to save the data to a file. The file format is the same as for the internal data gathering.

12.0 Control Loops

12.1 HXP Servo Loops

Chapter 14.0 is intended as reference for tuning the SingleAxis stage(s). Contact a Newport Engineering for tuning the Hexapod Group.

12.1.1 Servo structure and Basics

The HXP controller can be used to control a wide range of motion devices, which are categorized by the HXP as “positioners”. Within the structure of the HXP' firmware, a “positioner” is defined as an object with an associated profile (trajectory), a PID corrector, a motor interface, a driver, a stage and an encoder.

The general schematic of a positioner servo loop is below.

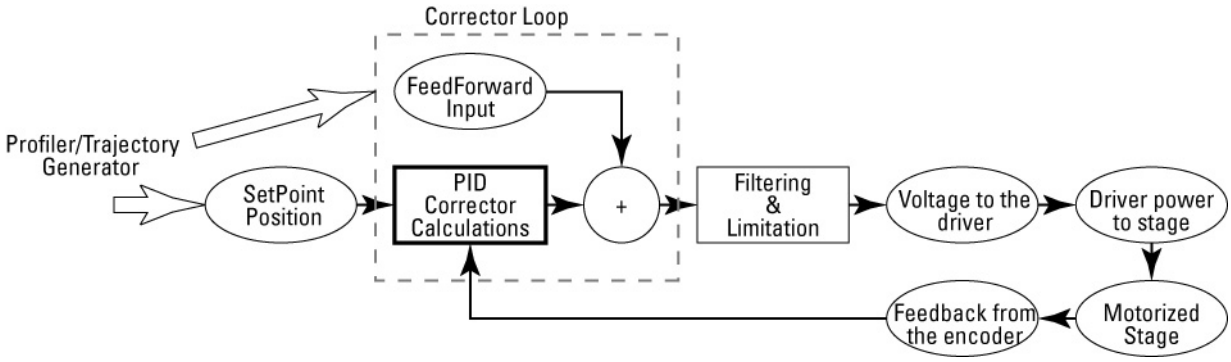


Figure 29: Servo structure and Basics.

The calculations done by the “servo loop” result in a voltage output from the controller that is applied to the driver, which can be either any of Newport's Universal drive modules or to an external driver through the HXP pass-through module. Depending on the corrector loop type selected, the level of this output voltage can be the result of two gain factors, the PID corrector and the FeedForward loop. The HXP has imbedded configuration files that provide optimized corrector loop settings for all Newport stages. Non-Newport stages may need to be assigned a specific corrector loop setting during the set-up process. In addition to the two main gain loops the HXP also adds filtering and error compensation parameters to this servo loop to improve system response and reliability.

The profiler (Trajectory Generator) within the controller calculates in real time, the position, velocity, and acceleration/deceleration that the positioner must follow to reach its commanded position (Setpoint Position). This profile is updated at Motion Profiler Rate.

The PID corrector then compares the SetpointPosition, as defined by the profiler, and the current position, as reported by the positioner's encoder, to determine the current following error. The PID corrector then outputs a value that the controller uses to maintain, increase or decrease the output voltage, which is applied to the driver. This loop is updated at the servo rate. The adjustment of the PID parameters allows users to optimize the performance of their positioner or system by increasing or decreasing the responsiveness of the output to increasing or decreasing following errors. Refer to the section 14.3 on PID tuning for more information and tips on PID tuning. The PID corrector loop and trajectory generation loop rates have been optimized to provide the highest level of precision. In most applications the critical control loop is the PID corrector since it has the most significant impact on positioning performance. Because of this, the PID loop is updated at a faster rate than the profiler cycle to improve profile execution and minimize following errors.

The Feed-Forward gain generates a voltage output to the driver that is directly proportional to the input. The purpose of this gain is to generate a movement of the

positioner as close as possible to the desired move that is independent of the encoder feedback loop. Adding this Feed-Forward gain can help reduce any encountered following errors and thus requires less compensation by the PID gain corrector. For example, if a driver and positioner respond to a constant voltage by moving at a constant speed, then feed forward input would be dictated by the SetpointSpeed.

The HXP stores standard Newport stage configuration files that can be used to quickly and easily develop the stage and system initialization (.ini) files. Below is an example of a typical stage and the type of DriverName, MotorDriverInterface and CorrectorType each is assigned. These standard Newport settings will be optimal for virtually every application and users would only need to modify their corrector loop parameters (Kp, Kd, Ki) to optimize positioner performance. Similar configurations can be adopted for non-Newport stages that are of similar motor driver types.

- Stages with high current (> 3 A) DC motor (RV, IMS) (with tachometer or back-emf estimation):

  DriverName: XPS-DRV01, 03

  ◇ ±10 V Input gives ±ScalingVelocity (stage velocity).

  ◇ Speed loop & Current loop configured by hardware.

  MotorDriverInterface: AnalogVelocity

  CorrectorType: PIDFFVelocity for Speed loop and PIDFFAcceleration for current loop.

- Stages with DC motor driven through a current loop (RGV) (no tachometer):

  DriverName: XPS-DRV02

  ◇ ±10 V Input gives ±ScalingAcceleration (stage acceleration).

  ◇ Current loop configured by hardware.

  MotorDriverInterface: AnalogAcceleration

  CorrectorType: PIDFFAcceleration

- Stages with low current (< 3 A) DC motor & tachometer (VP):

  DriverName: XPS-DRV01 in velocity mode.

  ◇ Input 1: ±10 V results in ±ScalingVelocity (theoretical stage velocity).

  ◇ Input 2: ±10 V results in ±ScalingCurrent (3 A).

  ◇ Speed loop programmable.

  MotorDriverInterface: AnalogVelocity

  CorrectorType: PIDFFVelocity

- Stages with low current (<3 A) DC motor, without tachometer (ILSCC type):

  DriverName: XPS-DRV01 in voltage mode.

  ◇ Input 1: ±10 V results in ±ScalingVoltage (48 V).

  ◇ Input 2: ±10 V results in ±ScalingCurrent (3 A).

  MotorDriverInterface: AnalogVoltage

  CorrectorType: PIDDualFFVoltage

- Stages with Stepper motor & Encoder (UTSPP, RVPE, ILSPP…):

  DriverName: XPS-DRV01 in stepper mode.

  ◇ Input 1: ±10 V results in ±ScalingCurrent in motor winding 1.

  ◇ Input 2: ±10 V results in ±ScalingCurrent in motor winding 2.

  MotorDriverInterface: AnalogStepperPosition

  CorrectorType: PIPosition

- Stages with Stepper motor & no encoder (TRA, SR50PP, PR50PP, MFAPP):

  DriverName: XPS-DRV01 in stepper mode.

  ◇ Input 1: ±10 V results in ±ScalingCurrent in motor winding 1.

  ◇ Input 2: ±10 V results in ±ScalingCurrent in motor winding 2.

  MotorDriverInterface: AnalogStepperPosition

  CorrectorType: NoEncoderPosition

These are just examples of available positioner associations in the HXP. The flexibility of positioner associations allows many other configurations to be developed to drive non-Newport positioners or other products. Before developing other configurations, the user must be aware that the main goal of creating these associations is to match the servo loop output to the appropriate driver input as stated by the manufacturer. For instance:

- The Corrector PIPosition is used when a constant voltage applied to a driver results in a constant position of the positioner (stepper motor, piezo, electrostrictive, etc.).
- Corrector PIDFFVelocity is used when a constant voltage applied to a driver results in a constant speed of the positioner (DC motor and driver board in speed loop mode).
- Corrector PIDFFAcceleration is used when a constant voltage applied to a driver results in a constant acceleration of the positioner (DC motor and driver board in current loop mode).
- Corrector PIDDualFFVoltage is used when a constant voltage applied to a driver results in a constant voltage applied to the motor (DC motor and driver board with direct PWM command).

12.1.2 HXP PIDFF Architecture

Corrector loops PIDFFVelocity, PIDFFAcceleration and PIDFFDualVoltage all use the same architecture as the PID corrector that is detailed below. PIPosition is a simplified version of this loop that is used to provide closed loop positioning via encoder feedback to stepper motor positioners.

12.1.2.1 PID Corrector Architecture

The PID corrector uses the following error (SetpointPosition – EncoderPosition) as its input and applies the sum of three correction terms (Kp, Kd and Ki) to determine the output.

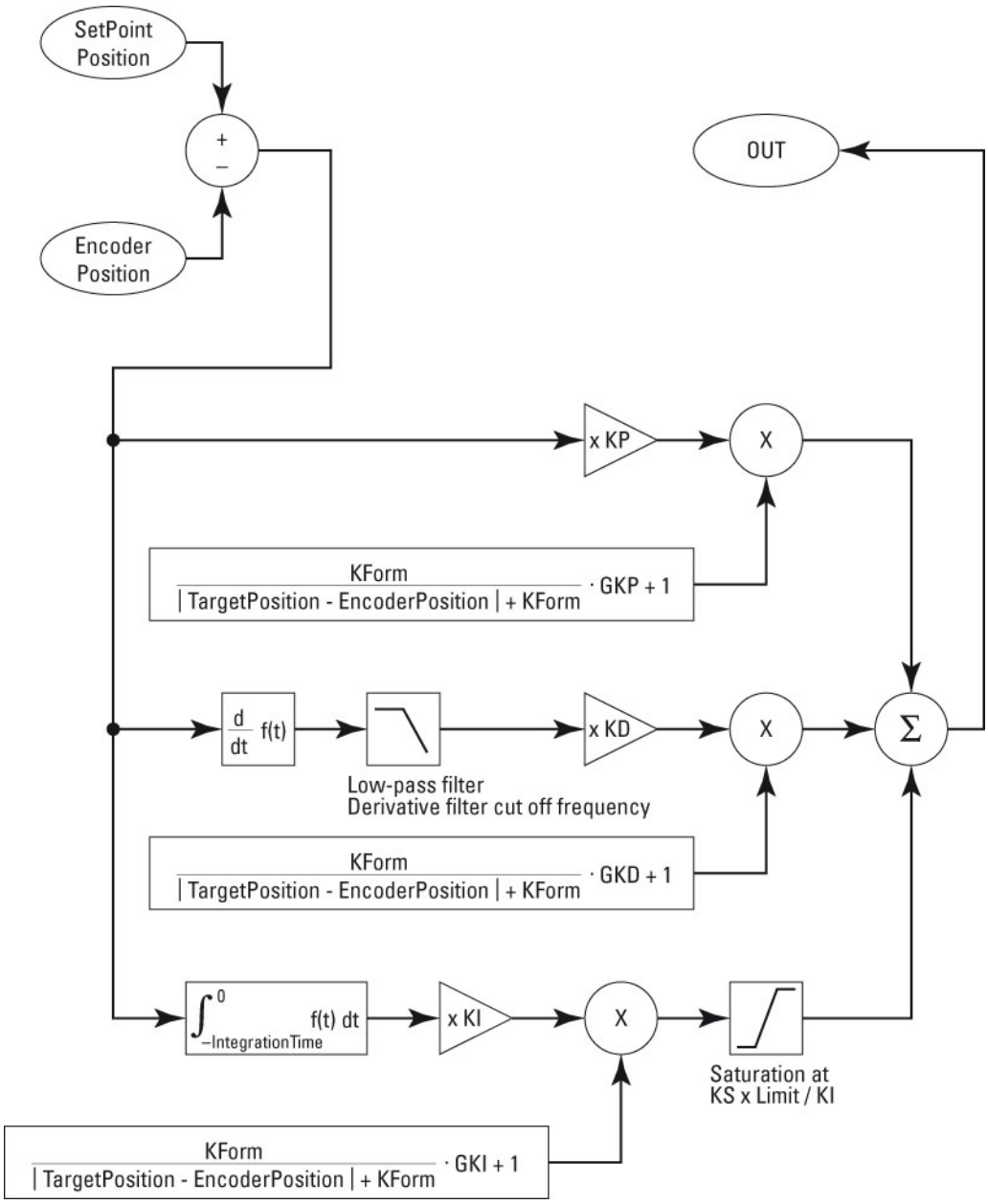


Figure 30: PID Corrector Architecture.

12.1.2.2 Proportional Term

The **Kp**, or proportional gain, multiplies the current following error of that servo cycle by the proportional gain value (Kp). The effect is to react immediately to the following error and attempt to correct it. Changes in position generally occur during commanded acceleration, deceleration, and in moves where velocity changes occur in the system dynamics during motion. As Kp is increased, the PID corrector will respond with a increased output and the error is more quickly corrected. For instance, if a positioner or group of positioners is expected to have small following errors, as is the case for small moves where overcoming static friction of the system is predominant, then the Kp may need to be increased to produce sufficient output to the driver. For larger moves, the following errors are generally larger and require lower Kp values to produce the desired output. Also note that for larger moves the kinetic friction of the system is generally

much lower than static friction and would generally require less correction gain than smaller moves. However, if Kp becomes too large, the mechanical system may begin to overshoot (encoder position > SetpointPosition), and at some point, it may begin to oscillate, becoming unstable if it does not have sufficient damping.

Kp cannot completely eliminate errors. However, since as the following error e, approaches zero, the proportional correction element, Kp x e, also approaches zero and results in some amount of steady-state error. For this reason other gain factors like Kd and Ki are required.

12.1.2.3 Derivative Term

The **Kd**, or derivative gain, multiplies the differential between the previous and current following error by the derivative gain value (Kd). The result of this gain is to stabilize the transient response of a system and can also be thought of as electronic damping of the Kp. The derivative acts as a gain that increases with the frequency of the variations of the following error:

$$\frac{d}{dt}\left[\sin(2\pi Fr\, t)\right] = 2\pi Fr \cos(2\pi Fr\, t)$$

The result is that the derived term becomes dominant at high frequencies, compared to the proportional and integral terms. For the same reason, the value of Kd is in most cases limited by high frequency resonance of the mechanics. This is why a low pass filter (cut off frequency = DerivativeFilterCutOffFrequency) is implemented in the derivative branch to limit excitation at high frequencies. Increasing the value of Kd increases the stability of the system. The steady-state error, however, is unaffected since the derivative of the steady-state error is zero.

These two gains alone can provide stable positioning and motion for the system. However to eliminate the steady state errors, an additional gain value must be used.

12.1.2.4 Integral Term

The Integral term **Ki** acts as a gain that increases when the frequency of the variations of the following error decrease:

$$\int\left[\sin(2\pi Fr\, t)\right] = \frac{1}{2\pi Fr}\sin(2\pi Fr\, t)$$

The result is that the integral term becomes dominant at low frequencies, compared to the proportional and derivative terms. The gain becomes infinite when frequency = 0. Even a very small following error will generate an infinite value of the integral term. The advantage of the integral term is that it will eliminate any steady-state following error. However, the disadvantage is that the integral term can reach values where the corrector is saturated causing the system to become unstable at the end of a move and cause the positioner to hunt or dither. To reduce this effect, two additional parameters are included in the PID corrector to help prevent these instabilities, Ks and Integration Time.

Ks

The saturation limit factor Ks permits users to limit the maximum value of Ki that is applied to the total PID corrector output. The Ks saturation limit can be set between 0 and 1, a typical setting is 0.5. As an example, at a setting of 0.5, the maximum output generated by the Ki term applied to the PID output would be 0.5 x the maximum set output. However, if the Ki gain factor output is less than 0.5 x the maximum set output, then the entire gain will be applied to the PID corrector. This maximum output is set within the section MotorDriverInterface in the stages.ini using the parameters AccelerationLimit, VelocityLimit or VoltageLimit. Refer to the Programmers manual for more information on this function.

Integration Time

The IntegrationTime is used to adjust the duration for integration of the residual errors. This can help in applications where large following errors can occur during motion. The

use of a small Integration Time value will limit the integration range to the latter parts of the move, avoiding the need of a large overshoot at the end of the move to clear the integrated following error value. The drawback is that the static error will be less compensated.

12.1.2.5 Variable Gains

In addition to the classical Kp, Ki, and Kd gain parameters, the XPS PID Corrector Loop also includes variable gain factors GKp, GKd, and GKi. These can be used to reduce settling time on systems that have nonlinear behavior or to tighten the control loop during the final segment of a move. For example, a positioner or stage with a high level of friction will have a response which is dependent on the size of the move: friction is negligible for a large move but becomes a predominant factor for small moves. For this reason, the required response of the system to reach the commanded position is not the same for small and large moves. The optimum value of PID parameters for small moves is very often higher than the optimum value for large moves. It is advantageous to modify PID settings depending on the move size. For users that do not need to make PID corrector adjustments (or prefer not to) benefit from the compensations provided by the variable gain correctors. This compensation is made automatically by the HXP variable gain corrector by applying a gain that is driven by the distance between the Target Position (position that must be reached at the end of the motion) and the Encoder Position. As shown in the figure below, when the distance to move completion is large, the total output gain from these parameters is fractional (the "Kform term" is fractional), but as the move size or distance to final position is small the Kform term approaches 1 and full GKx output is provided.

GKp = 10 Kp = 2
Target Position = 0
Encoder Position = -100 to 100

$$Kp_{(Kform,\ Encoder\ Position)} = \left[1 + GK \cdot \left(\frac{Kform}{|\ Target\ Position - Encoder\ Position\ | + Kform} \right) \right] \cdot Kp$$

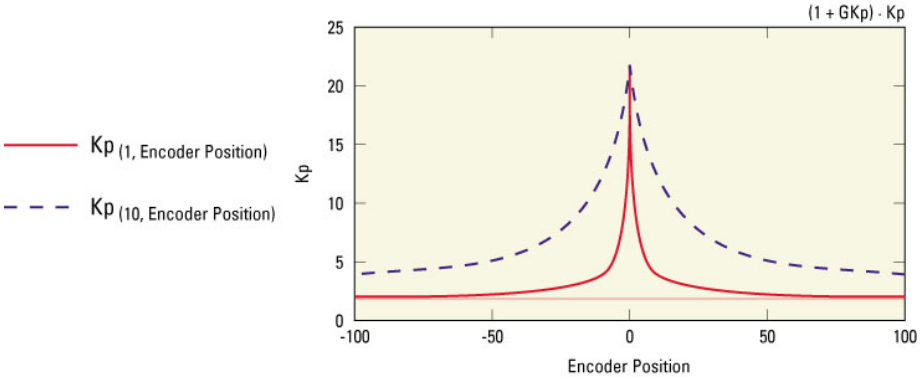


Figure 31: Variable Gains.

The parameter GKx is used to adjust the amplitude of the total output and the parameter Kform is used set how soon this Gkx is applied. As seen in the figure below, if a Kform of 1 is implemented, the GKx is not applied until the positioner is very close to its target position, in this case 0. But a Kform of 10 will implement the GKx much sooner and tighten the control of the loop further from the target position. This can be very effective when positioning high inertial loads or when very short settling times are critical. The default setting for the Kform parameter is 0 for all standard Newport stages.

12.2 Filtering and Limitation

In addition to the various PID correctors and calculations, filtering and limitation parameters also have the same structure for all the correctors (PIDFFVelocity, PIDFFAcceleration and PIDFFDualVoltage, etc).

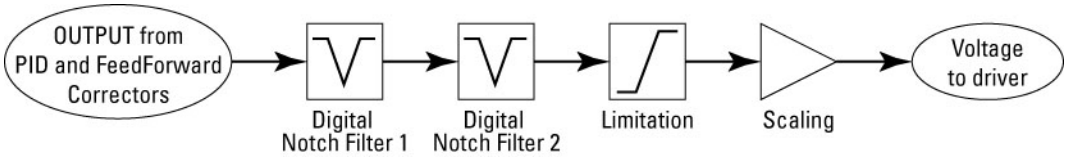


Figure 32: Filtering and Limitation.

The first section of the above diagram shows the succession of two digital notch filters. Each filter is defined by its central frequency (NotchFrequency), its bandwidth (NotchBandwidth) and its gain (NotchGain).

The gain, usually in the range of 0.01 to 0.1, is the value of the amplification of a signal at a frequency equal to the central frequency and the bandwidth is the range about the central frequency for which this gain is equal to a -3 db reduction.

Notch filters are typically used to avoid the instability of the servo loop due to the mechanic's natural frequencies, by lowering the gain at these frequencies. When they are implemented, these filters add some phase shift to the signal. This phase shift increases with the filter bandwidth and must remain small in the frequency range where the servo loop is active to maintain stability. The result is that notch filters are only effective at avoiding instabilities due to excessive and constant natural frequencies.

The last section of the diagram shows the limitation and scaling features. Scaling is used to transform units of position, speed or acceleration to a corresponding voltage. The Limitation factor is a safety that is used to limit the maximum voltage that can be applied to the driver to protect against any runaway or saturation situations that may occur.

12.3 Feed Forward Loops and Servo Tuning

12.3.1 Corrector = PIDFFVelocity

The PIDFFVelocity corrector should be implemented into applications where the positioner driver requires a "speed" input (constant voltage to the driver provides constant speed output to the positioner), using MotorDriverInterface = AnalogVelocity.

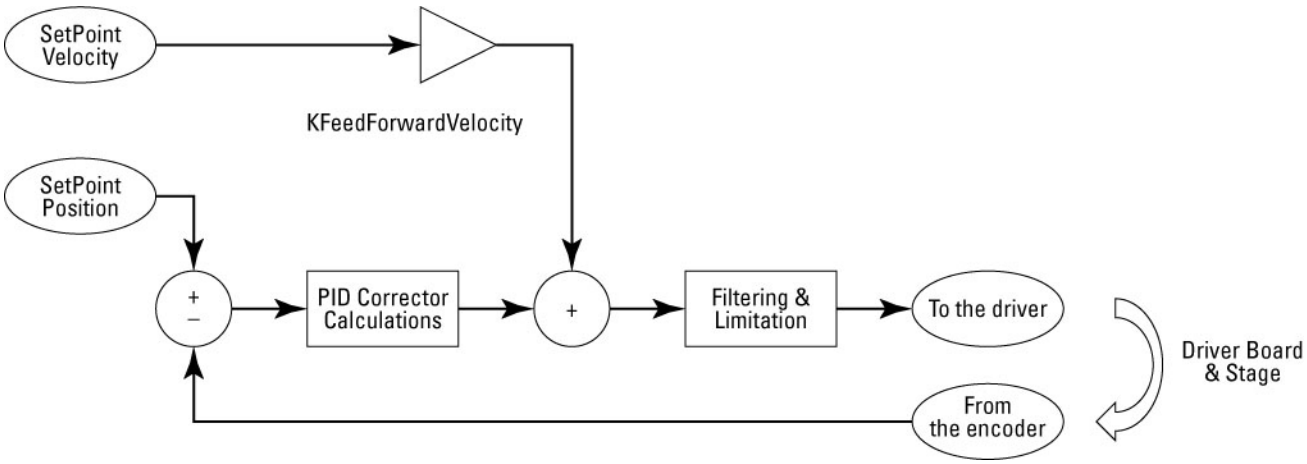


Figure 33: Corrector = PIDFFVelocity.

12.3.1.1 Parameters

FeedForward Method:

- Velocity
- KFeedForwardVelocity is a gain that can be applied to this feed forward.
- When the system is used in open loop, the PID output is not applied and the feed forward gain is set to 1 (the entire output of the controller is FF gain).

PID corrector:

- Total output of the PID is a speed (units/s), so:

  Kp is given in 1/s.

  Ki is given in $1/s^2$.

  Kd has no unit.

Filtering and Limitation:

- ScalingVelocity (units/s) is the theoretical speed resulting from a 10 V input to the driver.
- VelocityLimit (units/s) is the maximum speed that can be commanded to the driver.

12.3.1.2 Basics

For a “perfect system” (no friction, all performance factors known, no following errors), a KFeedForwardVelocity value of 1 will generate the exact amount of output required to reach the TargetPosition.

The Kd parameter is generally redundant when using the speed loop of the driver and is usually set to zero, but a higher value can be used to improve the “tightness” of the speed loop.

The proportional gain Kp drives the cut-off frequency of the closed loop.

Due to the integration of the speed command in a position by the encoder, the overall gain of the proportional path at a given frequency Frq is equal to $Kp/2\pi Frq$. This gain is equal to 1 at Frq $P = Kp/2\pi$ (close to the cut-off frequency).

This frequency must remain lower than the cut-off frequency of the speed loop of the driver and lower than the mechanic’s natural frequencies to maintain stability.

The integral gain Ki drives the capability of the closed loop to overcome perturbations and to limit static error.

Due to the integration of the speed command in a position by the stage encoder, the overall gain of the integral path at a given frequency Frq is:

$$\text{Gain} = \frac{\text{Ki}}{(2 \cdot \pi \cdot \text{Frq})^2}$$

This gain is equal to one at FrqI:

$$\text{FrqI} = \frac{1}{2 \cdot \pi} \cdot \sqrt{\text{Ki}}$$

This frequency FrqI must typically remain lower than the frequency FrqP of the proportional path to keep the stability of the servo loop.

12.3.1.3 Methodology of Tuning PID's for PIDFFVelocity Corrector (DC motors with or without tachometer)

1. Verify the speed in open loop (adjustment done using ScalingVelocity).
2. Close the loop, set Kp, increase it to minimize following errors to the level until oscillations/vibrations start during motion, then decrease Kp slightly to cancel these oscillations.
3. Set Ki, increase it to limit static errors and improve settling time until the appearance of overshoot or oscillation conditions. Then reduce Ki slightly to eliminate these oscillations.
4. Kd is generally not needed but it can help in certain cases to improve the response when the speed loop of the driver board is not efficient enough.

Note

To set the corrector parameters (loop type, Ki, Kp, Kd,...), use the following functions (refer to Programmer's Manual for details):

- **CorrectorType = PIDFFVelocity : PositionerCorrectorPIDFFVelocitySet(...)**
- **CorrectorType = PIDFFAcceleration: PositionerCorrectorPIDFFAccelerationSet(...)**
- **CorrectorType = PIDDualFFVoltage: PositionerCorrectorPIDDualFFVoltageSet(...)**
- **CorrectorType = PIPosition: PositionerCorrectorPIPositionSet(...)"**

12.3.2 Corrector = PIDFFAcceleration

The PIDFFAcceleration must be used in association with a driver having a torque input (constant voltage gives constant acceleration), using MotorDriverInterface = AnalogAcceleration. (AnalogSin60Acceleration, AnalogSin90Acceleration, AnalogSin120Acceleration, AnalogDualSin60Acceleration, AnalogDualSin90Acceleration or AnalogDualSin120Acceleration).

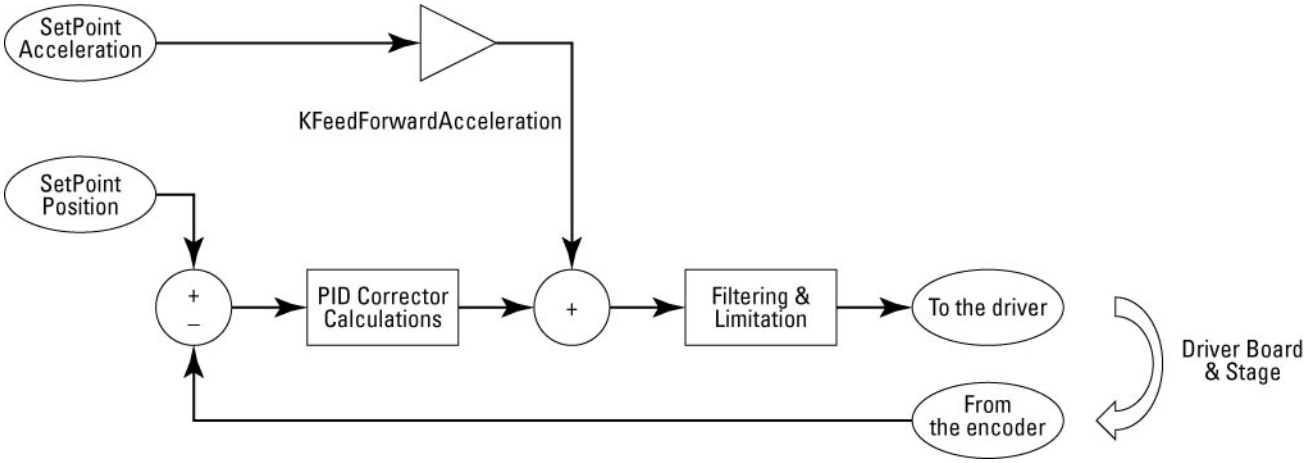


Figure 34: Corrector = PIDFFAcceleration.

12.3.2.1 Parameters

FeedForward method:

- A feed forward in acceleration is used.
- KFeedForwardAcceleration is a gain that can be applied to this feed forward.
- When the system is used in open loop, the PID output is cut and the feed forward gain is set to 1.

PID corrector:

- Output of the PID is an acceleration value in units/s^2.

  Kp is given in $1/s^2$.

  Ki is given in $1/s^3$.

  Kd is given in 1/s.

Filtering and Limitation:

- ScalingAcceleration (units/s^2) is the theoretical acceleration of the stage resulting from a 10 V input to the driver (depends on the stage payload).
- AccelerationLimit (units/s^2) is the maximum acceleration allowed to be commanded to the driver.

12.3.2.2 Basics

The derivative term Kd drives the cut-off frequency of the closed loop and must be adjusted first (the loop will not be stable with only Kp).

Due to the double integration of the acceleration command in a position by the stage encoder, the overall gain of the derivative path at a given frequency Frq is equal to $Kd/2\pi Frq$. This gain is equal to one at $FrqD = Kd/2\pi$ (close to servo loop cut-off frequency). This frequency must remain lower than the cut-off frequency of the current loop of the driver and lower to mechanical natural frequencies to keep the stability.

The proportional gain Kp drives mainly the capability of the closed loop to overcome perturbations at medium frequencies and to limit following errors. Due to the double integration of the acceleration command in a position by the stage encoder, the overall gain of the proportional part at a given frequency Frq is:

$$\mathrm{Gain} = \frac{\mathrm{Kp}}{(2 \cdot \pi \cdot \mathrm{Frq})^2}$$

This gain is equal to one at FrqP:

$$\mathrm{FrqP} = \frac{1}{2 \cdot \pi} \cdot \sqrt{\mathrm{Kp}}$$

This frequency FrqP must remain lower than the frequency FrqD of the derivative part to keep the stability.

The integral gain Ki drives the capability of the closed loop to overcome perturbations at low frequencies and to limit static error.

Due to the double integration of the acceleration command in a position by the stage encoder, the overall gain of the integral part at a given frequency Frq is:

$$\mathrm{Gain} = \frac{\mathrm{Ki}}{(2 \cdot \pi \cdot \mathrm{Frq})^3}$$

This gain is equal to one at FrqI:

$$\mathrm{FrqI} = \frac{1}{2 \cdot \pi} \cdot \mathrm{Ki}^{\frac{1}{3}}$$

This frequency FrqI must remain lower than the frequency FrqP of the proportional part to keep the stability.

12.3.2.3 Methodology of Tuning PID's for PIDFFAcceleration Corrector (direct drive DC motors)

1. Verify the AccelerationFeedForward in open loop (adjustment done using ScalingAcceleration).

   Close the loop, set Kd, increase it to minimize following errors until vibrations appear during motion.
2. Decrease Kd to eliminate oscillations.
3. Set Kp, increase it to minimize following errors until the appearance of oscillations, decrease it to eliminate oscillations.
4. Set Ki, increase it to limit static errors and settling time until the appearance of overshoot/oscillations.

Note

To set the corrector parameters (loop type, Ki, Kp, Kd,...), use the following functions (refer to Programmer's Manual for details):

- **CorrectorType = PIDFFVelocity : PositionerCorrectorPIDFFVelocitySet(...)**
- **CorrectorType = PIDFFAcceleration: PositionerCorrectorPIDFFAccelerationSet(...)**
- **CorrectorType = PIDDualFFVoltage: PositionerCorrectorPIDDualFFVoltageSet(...)**
- **CorrectorType = PIPosition: PositionerCorrectorPIPositionSet(...)"**

12.3.3 Corrector = PIDDual FFVoltage

The PIDDualFFVoltage must be used in association with a driver having a voltage input (constant voltage gives constant motor voltage), using MotorDriverInterface = AnalogVoltage.

Can also be used in velocity or acceleration command.

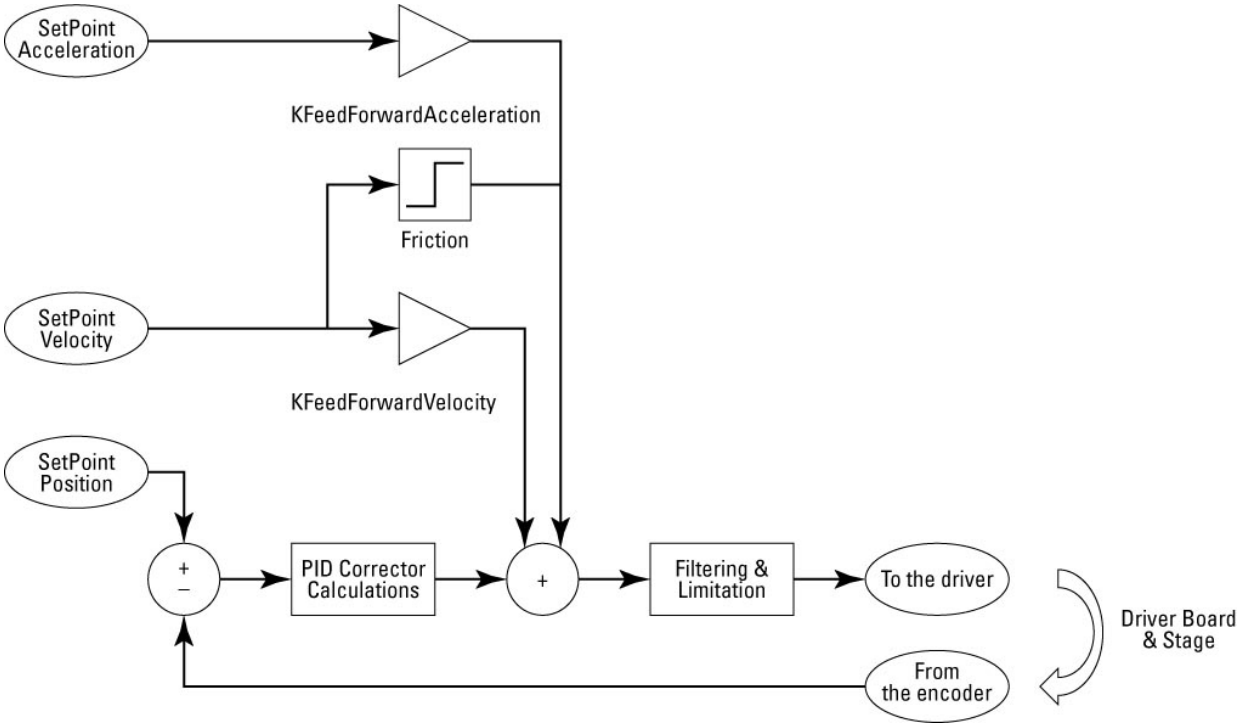


Figure 35: Corrector = PIDDual FFVoltage.

12.3.3.1 Parameters

FeedForward method:

- 3 feed forwards are used: Speed, Acceleration and Friction.
- KFeedForwardAcceleration is a gain that can be applied to the feed forward in acceleration.
- KFeedForwardVelocity is a gain that can be applied to the feed forward in velocity.
- Friction is a value which is applied with the sign of the velocity.
- When the system is used in open loop, the PID output is cut and only one feed forward in velocity is applied with the gain defined by KFeedForwardVelocityOpenLoop.

PID corrector:

- Output of the PID is a voltage.

  Kp is given in V/unit.

  Ki is given in V/unit/s.

  Kd is given in V/s/unit.

Filtering and Limitation:

- ScalingVoltage is the theoretical motor voltage resulting from a 10 V input on the driver (48 V).
- VoltageLimit (volts) is the maximum motor voltage allowed to be commanded to the driver.

Refer to the XPS-Q8 Configuration Wizard Document for a detailed explanation.

12.3.3.2 Basics

The PIDDualFFVoltage corrector can be seen as a mix between the PIDFFVelocity and PIDFFAcceleration correctors. It is difficult to give a precise picture of this behavior which depends a lot on the response of the stage (speed and acceleration versus motor voltage).

12.3.3.3 Methodology of Tuning PID's for PIDDualFF Corrector (DC motors with tachometers)

1. Adjust KFeedForwardVelocityOpenLoop to optimize the fidelity of the speed at high speed.
2. Close the loop using the same value for KFeedForwardVelocity, set Kp, increase it to minimize following errors until oscillations/vibrations appears during motion, decrease Kp to eliminate oscillations.
3. Set Kd, increase until oscillations/vibrations appear during motion, and decrease it to eliminate oscillations.
4. Increase Ki to cancel static error and minimize settling time until appearance of overshoot/oscillations.

12.3.4 Corrector = PIPosition

PIPosition corrector can be used with AnalogStepperPosition or AnalogPosition interface.

The AnalogPosition interface is to be used with a driver having a position input (example = piezo driver).

The AnalogStepperPosition interface is to be used with a driver having two sine and cosine current inputs (constant voltage gives constant currents in motor windings so position is constant).

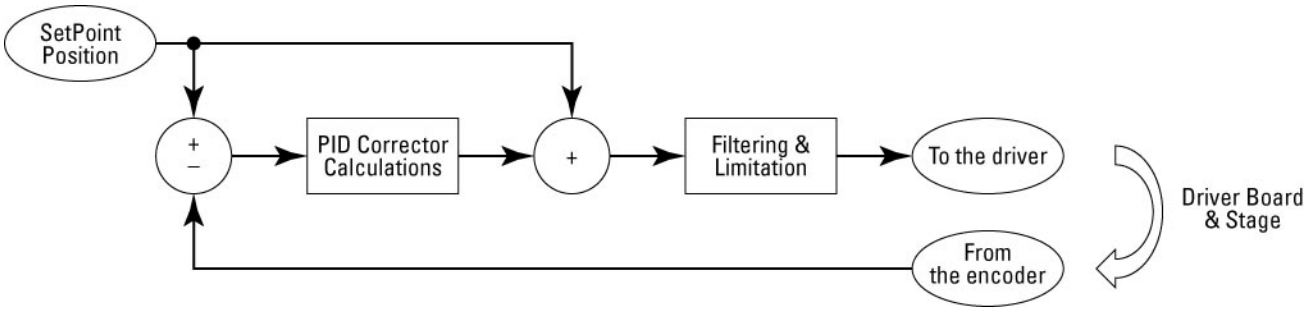


Figure 36: Corrector = PIPosition.

12.3.4.1 Parameters

FeedForward:

- One feed forward in position. No adjustable gain.
- When the system is used in open loop, the PI output is cut and the feed forward in position is applied.

PI corrector:

- Output of the PI is a position.

  Kp has no units.

  Ki is given in 1/s.

12.3.4.2 Basics & Tuning

In most cases, only Ki is needed to correct static errors.

The overall gain of the integral part of the servo loop at a given frequency Frq is:

$$\text{Gain} = \frac{\text{Ki}}{2 \cdot \pi \cdot \text{Frq}}$$

This gain is equal to one at:

$$\text{FrqI} = \frac{\text{Ki}}{2 \cdot \pi}$$

13.0 Introduction to HXP Programming

For advanced applications and repeating tasks, it is usually better to sequence different functions in a program rather than executing them manually via the web site interface. Motion processes can be written in different ways, but essentially can be distinguished between host-managed processes, processes controlled from a PC with communication to the HXP via the Ethernet TCP/IP interface and HXP-managed processes, processes controlled directly by the HXP controller via a TCL script.

Host-managed processes

Host-managed processes are recommended for applications that require a lot of data management or a lot of digital communication with other devices other than the HXP controller. In this case, it is more efficient to control the process from a dedicated program that runs on a PC and which sends (and gets) information to (and from) the HXP controller via the Ethernet TCP/IP communication interface. Communication to the HXP controller can be established from almost any PC and is independent of the PC's operating system (Windows, Linux, Unix, Mac OS…) and programming language (LabView, C++, C, VisualBasic, Delphi, etc.). The HXP controller supports the development of host-managed processes with a Windows communication DLL, a complete set of LabView drivers and a number of example programs in C++, VisualBasic and LabView. A few basic examples are provided in this section. For more details, please refer to the Software Drivers Manual.

HXP-managed processes (TCL)

The HXP controller is also capable of controlling processes directly using TCL scripts. TCL stands for Tool Command Language and is an open-source string-based command language. With only a few fundamental constructs, it is very easy to learn and it is almost as powerful as C. Users of HXP can use TCL to write complete application code and HXP allows them to include any function to a TCL script. When developed, the TCL script can be executed in real time on the motion controller in the background, utilizing time that the controller does not need for servo or communication. Multiple TCL programs run in a time sharing mode. To learn more about TCL, refer to the TCL Manual which is accessible from the web site of the HXP controller.

The advantage of HXP-managed processes compared to host-managed processes is faster execution and better synchronization in many cases without any time taken from the communication link. HXP-managed processes or sub-processes are particularly valuable with repeating tasks, tasks that run in a continuous loop, and tasks that require a lot of data from the HXP controller. Examples include: anti-collision processes, processes that utilize security switches to stop motion when stages are in danger of collision; tracking, auto-focusing or alignment processes, processes that use external data inputs to control the motion; or custom initialization routines, processes that must constantly be executed during systems use.

The HXP controller has real-time multi-tasking functionality, and with most applications it is not only a choice between a host-managed or an HXP-managed process, but also a recognition of splitting the application into the right number of sub-tasks, and defining the most efficient process for each sub-task. An efficient process design is one of the main challenges with today's most complex and critical applications in terms of time and precision. It is recommended to spend a lot of thought to the proper definition of the best process approach.

The aim of this section is to provide a brief introduction to the different ways of HXP programming. This section, however, cannot address all details. For further information, refer to the TCL and the software drivers manual of the HXP controller which are accessible via the HXP web site interface.

13.1 TCL Generator

The TCL generator provides a convenient way of generating simple executable TCL scripts. These scripts may serve also as a good place to start for the development of more complex scripts.

The TCL generator is accessible from the terminal menu of the HXP web site. Pressing the TCL generator button generates a TCL script that includes the commands previously executed and listed in the Command history list. Note that the order in the TCL script is the same as executed and inverse to the order of the Command history list. The name of the TCL script is History.tcl and is stored in the ..\Admin\Public\Scripts folder of the controller.

Example

The TCL generator provides a convenient way of generating simple executable TCL scripts. These scripts may serve also as a good place to start for the development of more complex scripts.

The TCL generator is accessible from the terminal menu of the HXP web site. Pressing the TCL generator button generates a TCL script that includes the commands previously executed and listed in the Command history list. Note that the order in the TCL script is the same as executed and inverse to the order of the Command history list. The name of the TCL script is History.tcl and is stored in the ..\Admin\Public\Scripts folder of the controller.

Example

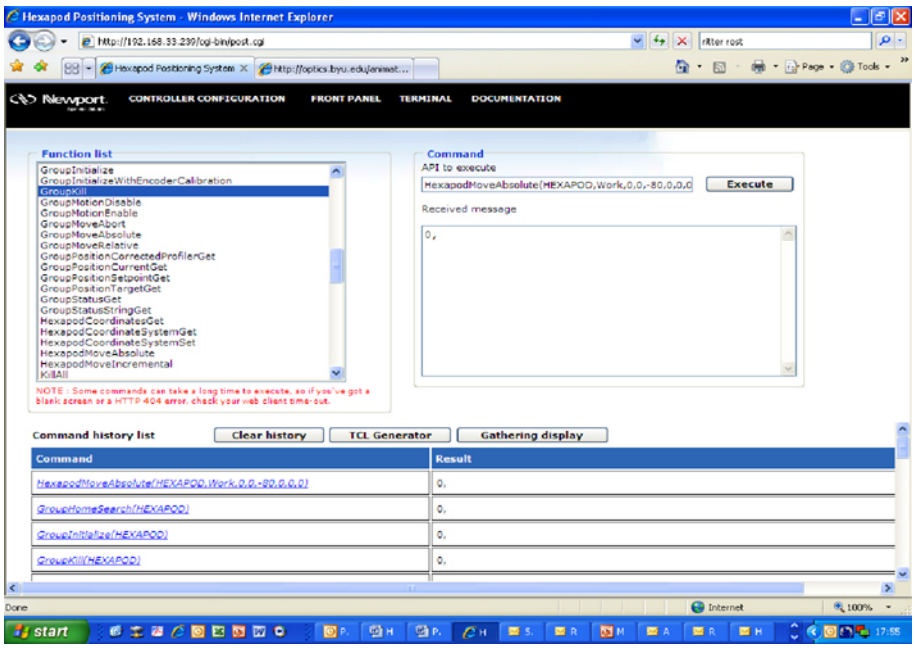
The following functions were executed:

GroupKill (HEXAPOD)

GroupInitialize(HEXAPOD)

GroupHomeSearch(HEXAPOD)

HexapodMoveAbsolute(0, 0, -80, 0, 0, 0)



To execute the script, use the function TCLScriptExecute(History.tcl, task1, 0).

In this example, after initializing and homing, the TCL script moves the Hexapod HEXPOAD to the position (0, 0, -80, 0, 0, 0)

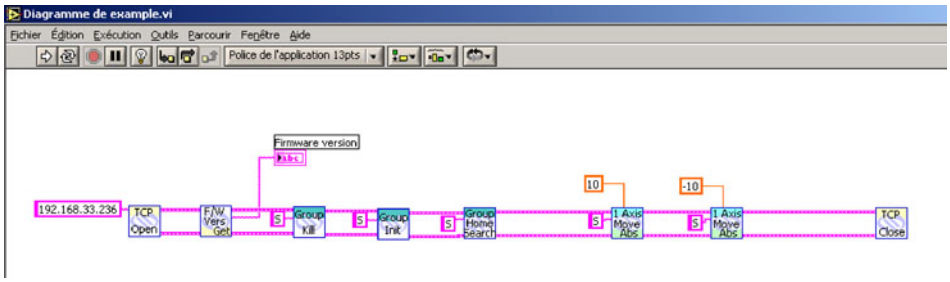
NOTE

Selecting the function TCLScriptExecute() from the terminal menu opens a drop-down list for the available TCLFileNames. However, this list is limited to 100 entries.

To learn more TCL programming, refer to the TCL Manual accessible from the documentation menu of the HXP web site. The TCL manual provides a complete description of all TCL commands and some more complex examples of TCL scripts.

13.2 LabView VIs

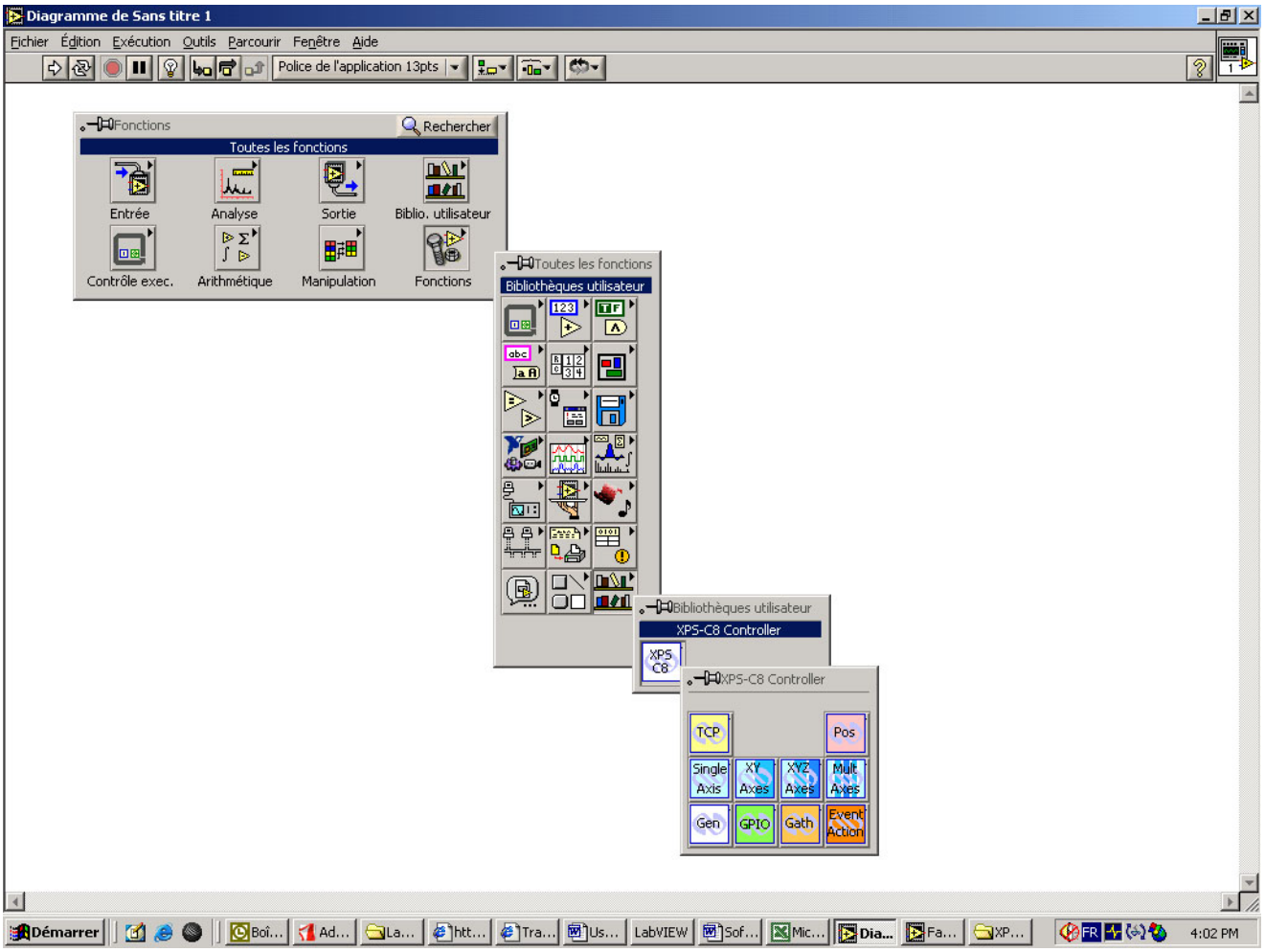
LabView is one of the most popular programming languages for the HXP controller. Newport provides a complete list of LabView drivers for the HXP controller, which means that LabView VIs exist for all HXP commands. In this section, a simple LabView application was developed that sequentially sends a number of commands to the HXP. The final application is illustrated as follow:



To use the HXP LabView drivers, the library files from the *../Admin/Public/Drivers/LabView/HXP* Controller folder of the HXP controller must be copied to the folder *Program Files/National Instruments/LabVIEW6.1/user.lib/HXP* Controller of your host computer.

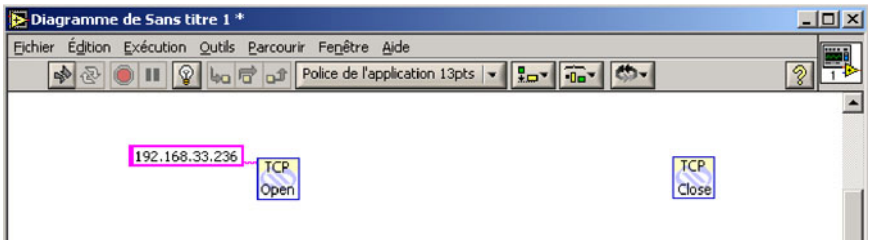
Start the LabView software and open an empty VI.

All drivers are located in the menu: *Window->Functions Pallet->Functions-> User library->HXP*. The drivers are classified in groups: TCP, General, Single axis, XY axes, XYZ axes, Multiple axes, Positioner, GPIO, Gathering and Events actions).

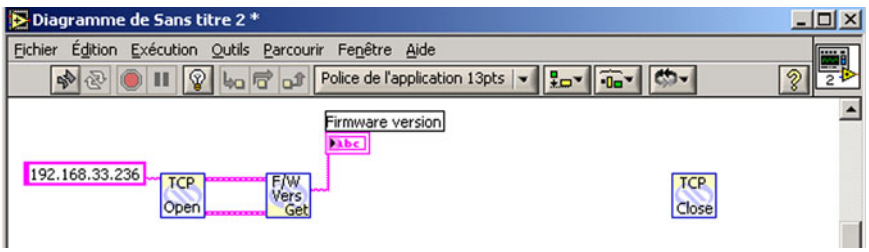


All LabView routines must begin with a *TCP Open* and must end with a *TCL Close*. If the TCP connection is not well managed, there will be no communication with the HXP. Then, add a constant chain (*Window->Functions Pallet->Functions->Chain*) to indicate the IP address of the controller and link it to the TCP Open driver.

When passing the cursor on the in/out panes of the driver, the required information is displayed.

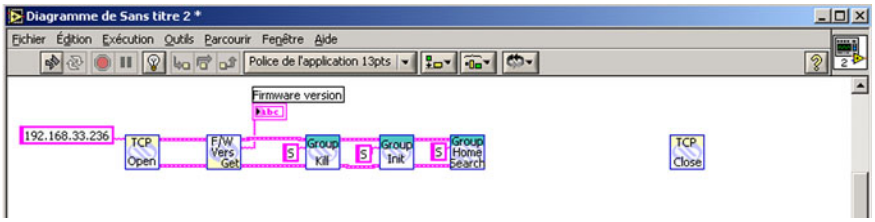


Add a Firmware Version Get from the general group and associate an indicator to its second output. Link TCP Open to Firmware Version Get.

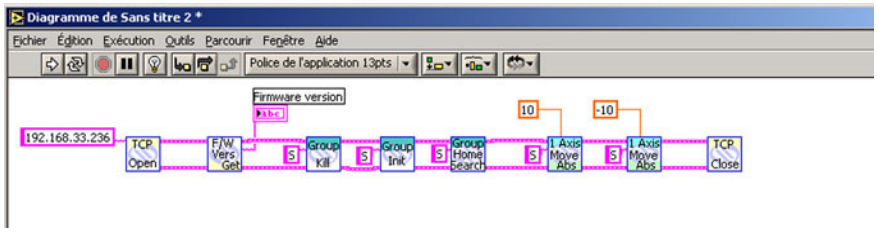


Then, add the initialization part: Add Group Kill, Group Initialize and Home search drivers from the single axis group and indicate the name of the group to each driver.

All drivers, except TCP Open and TCP Close, require a connection ID in (top left pane), a connection ID out (top right pane), an error in (bottom left pane) and an error out (bottom right.

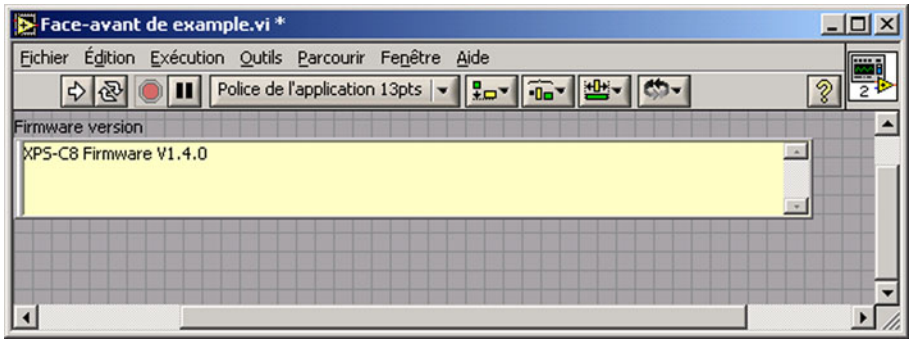


Add two Group move absolutes from the single axis group and indicate the name of the group. On the pane Target position, add a constant in which the desired absolute positions is written. Finally, link the drivers together and link the last move to the TCP Close.



To test the program, press the button  .

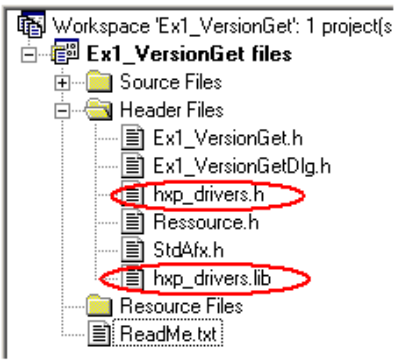
This example displays the firmware version of the HXP controller and executes a motion of the single axis group from 0 to +10, and then to –10 units.

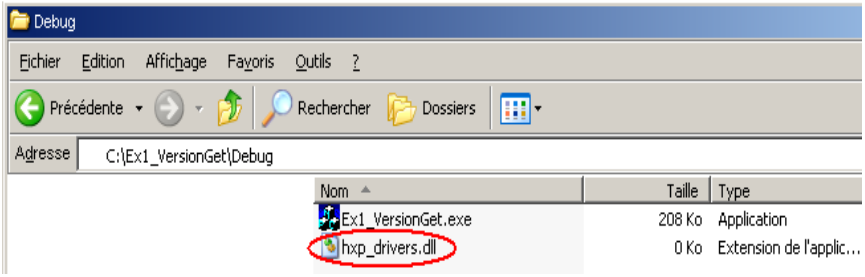


To learn more the HXP LabView drivers and their use, refer to the Software drivers manual accessible from the documentation menu of the HXP web site.

13.3 DLL Drivers

A DLL simplifies function calls from most programming languages. The DLL of the HXP controller is located in the ..Admin/Public/Drivers/dll folder of the HXP controller. The files hxp_drivers.h and hxp_drivers.lib must be copied to the project folder and the file hxp_drivers.dll to the folder of the executable file.





Once these files are added, for instance to a C++ project, the prototypes of the functions can be called in the program with the respective syntax of the functions (parameters number, type…). The file hxp_drivers.h can be opened to see the list of the available functions and their prototypes.

For instance, the prototype of the function FirmwareVersionGet is as follow:

```
DLL int __stdcall FirmwareVersionGet (int SocketIndex, char * Version);
```

It requires two arguments (int and char*).

Example of C++ sequence

```
char  buffer [256] = {'\0'};
char pIPAddress[15] = {"192.168.33.236"};
int  nPort = 5001;
double dTimeOut = 60;
int  SocketID = -1;

// TCP / IP connection
SocketID = TCP_ConnectToServer(pIPAddress, nPort, dTimeOut);
if (SocketID == -1)
{
  sprintf (buffer, "Connection to @ %s, port = %ld failed\n", pIPAddress,
nPort);
  AfxMessageBox (buffer, MB_ICONSTOP);
}
else
{
  AfxMessageBox("Connected to target", MB_ICONINFORMATION);

  // Get controller version
FirmwareVersionGet (SocketID, buffer); // Get controller version
AfxMessageBox (buffer, MB_ICONINFORMATION);

  // TCP / IP disconnection
TCP_CloseSocket(SocketID);  // Close Socket
AfxMessageBox("Disconnected from target", MB_ICONINFORMATION);
}
```

This example opens a TCP connection, gets the firmware version of the HXP controller and closes the connection. The execution is displayed in message boxes:

To learn more about the DLL prototypes, refer to the *Programmer's Manual*, accessible from the web site interface of the HXP controller. All DLL prototypes are described there.

The *Software drivers manual*, also accessible from the HXP web site interface, provides further information about the use of the DLL and additional C++ programming examples.

13.4 Running Processes in Parallel

TCP provides a reliable, point-to-point communication channel that client-server applications on the Internet use to communicate with each other. To communicate over TCP, a client program and a server program establish a connection to one another. Each program binds a socket to its end of the connection. To communicate, the client and the server both read from and write to the socket that binds the connection.

Sockets are interfaces that can "plug into" each other over a network. Once "plugged in", the connected programs can communicate.

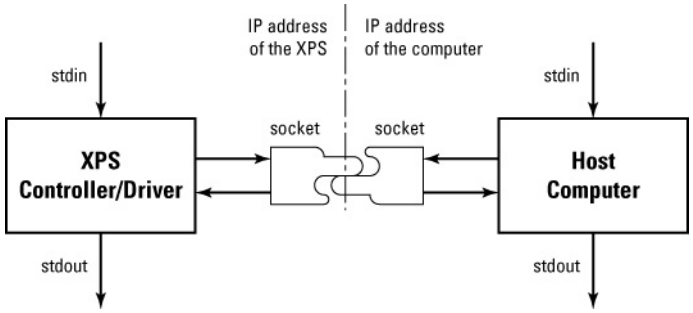


Figure 37: Running Processes in Parallel.

HXP uses blocking sockets. In other words, the programs/commands are "blocked" until the request for data has been satisfied. When the remote system writes data on the socket, the read operation will complete it and write it in the received message window of the Terminal menu ('0' if command has been well executed or the error number in case of an error). That way, commands are executed sequentially as each command always waits for a feedback before allowing execution of the next function. The main benefit of using this type of socket is that an execution acknowledgement is sent to the host computer with each function. In case of any error, it allows an exact diagnostic, which function has caused the error. It also allows a precise sequential process execution. On the other hand, more functions could be sent in parallel using non-blocking sockets. However, the drawback is that it is almost impossible to diagnose which function has caused an error.

To execute several processes in parallel, for instance to request the current position during a motion and other data simultaneously, it is possible to communicate to the HXP controller via different sockets. The HXP controller supports a maximum number of 30 parallel opened sockets. The total number of open communication channels to the HXP controller, be it via the website, TCL scripts, a LabView program, or any other program can not be larger than 30.

User's who prefer not to use blocking sockets, or whose programming languages don't support multiple sockets, such as Visual Basic versions prior to version .Net, can disable the blocking feature by setting a low TCPTimeOut value, 20 ms for instance. In

this case, the HXP will unblock the last socket after the TCPTimeOut time. However, this method loses the ability to pinpoint which commands were properly executed.

Examples of the use of parallel sockets

The following examples illustrate how to open several sockets via the web site interface, TCL scripts, LabView VIs and C++ programs.

Web site interface

The simplest way to open several sockets in parallel is to open several windows on the IP address of the controller. This is completely transparent to the user. Two or more groups of stages can be commanded for instance from two terminal menus at the same time to execute different motions (multitasking).

TCL scripts

A TCL script is carried out sequentially: the commands are executed one by one following the order they are written in the script. Consequently, there is no great interest to open several sockets in a single TCL script.

However, it is possible to start a TCL script from another TCL script. That way, as many sockets and parallel processes can be started in parallel as needed. Below is an example with 3 open sockets:

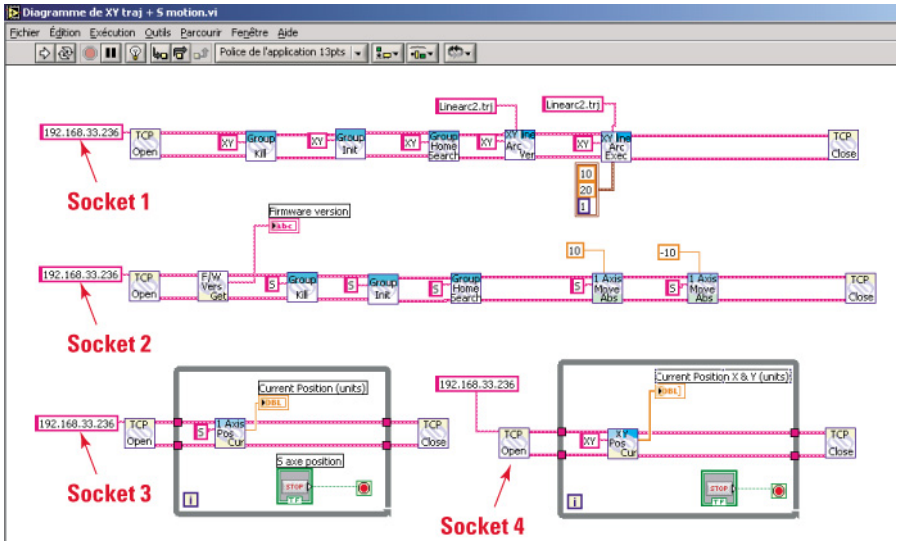
```
#####################
# TCL program : GEN #
#####################

set TimeOut 10
set code 0
set Prog1 “ProgRV.tcl”
set Task1 “Task1”
set Prog2 “ProgXY.tcl”
set Task2 “Task2”

# open TCP socket
set code [catch “OpenConnection $TimeOut socketID”]

if {$code == 0} {
    puts stdout “ProgGen : TCP_ConnectToServer OK => $code  ID =
$socketID”          <– Socket 1

set code [catch “TCLScriptExecute $socketID $Prog1 $Task1 0”]
puts stdout “ProgGen : TCLScriptExecute => error = $code”
                    <– Socket 2

set code [catch “TCLScriptExecute $socketID $Prog2 $Task2 0”]
puts stdout “ProgGen : TCLScriptExecute => error = $code”
                  <– Socket 3

# close TCP socket
set code [catch “TCP_CloseSocket $socketID”]
puts stdout “ProgGen : TCP_CloseSocket => $code  ID = $socketID”

} else  {
   puts stdout “ProgGen : TCP_ConnectToServer NOT OK => $code”
}
```

NOTE

Socket 2 and Socket 3 are not opened by the TCLExecuteScript function, but we supposed these scripts open some sockets on their own.

LabView VIs

In a VI file, several processes can easily be created, all beginning with a TCP Open and all finishing with a TCP Close. Each TCP Open will open its own socket. Shown below is a simple VI that opens 4 sockets at the same time.



C++ program

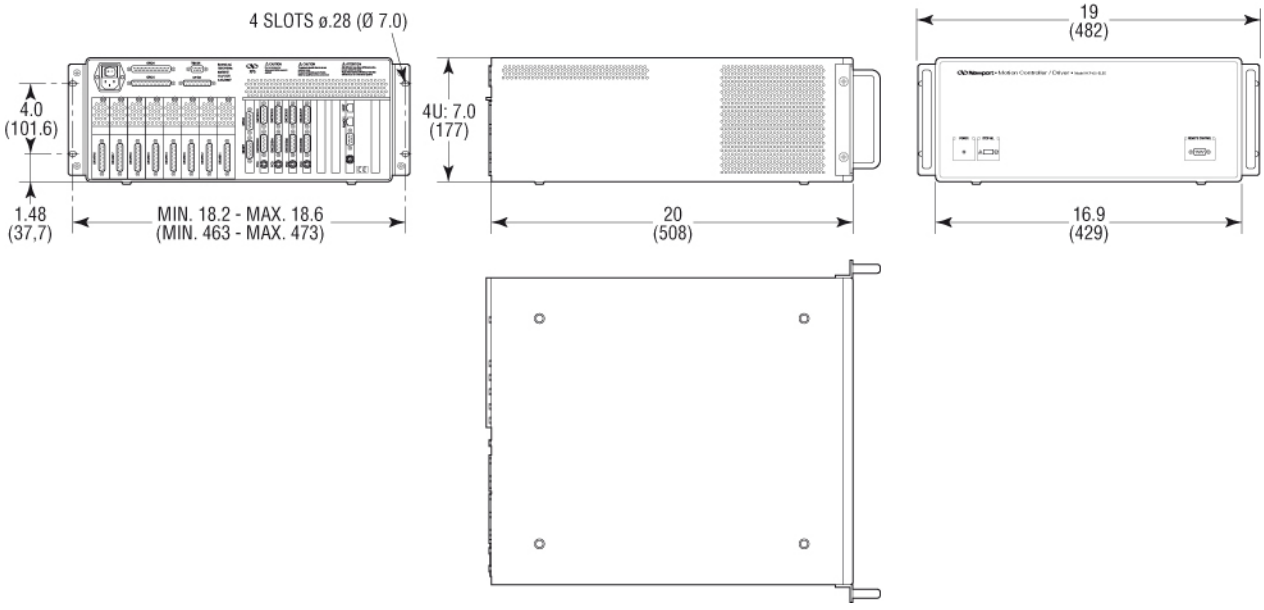
A C++ program is executed sequentially. Even if it calls many functions, they are always executed one by one following the order they are written. In order to open several sockets for multitasking, the C++ multithreading functionality must be used.

The HXP driver DLL allows a maximum number of 100 simultaneously opened sockets. One HXP controller supports a maximum number of 30 simultaneously opened sockets, but a program could control several HXP controllers.

Appendix

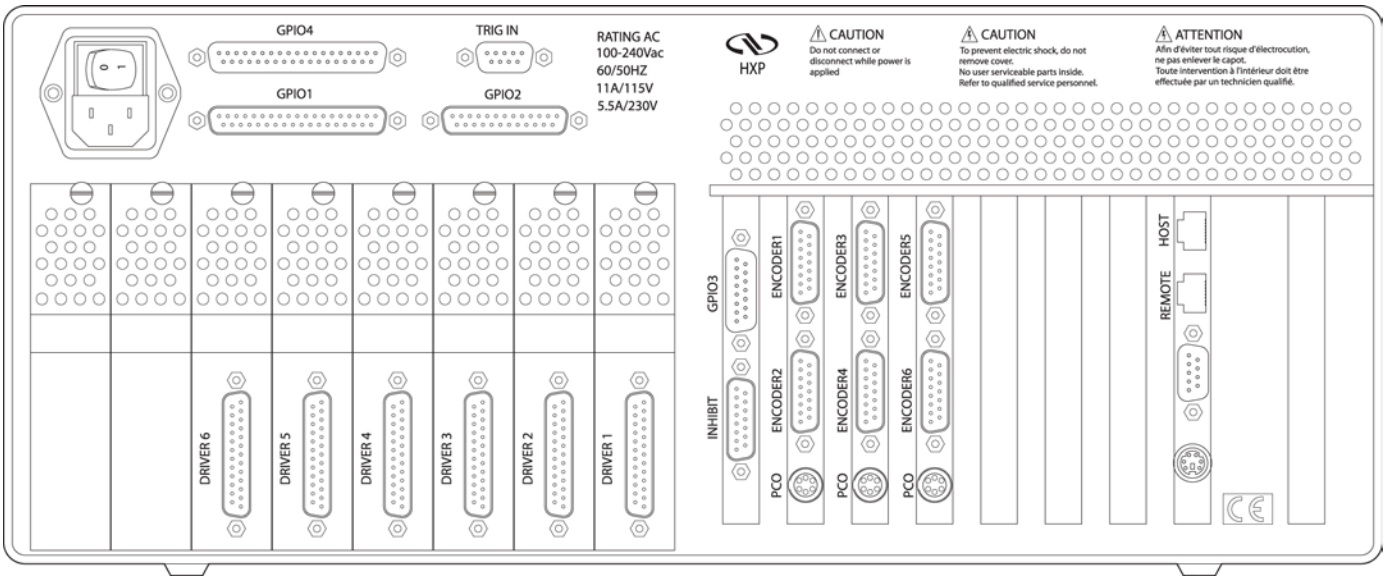
14.0 Appendix A: Hardware

14.1 Controller

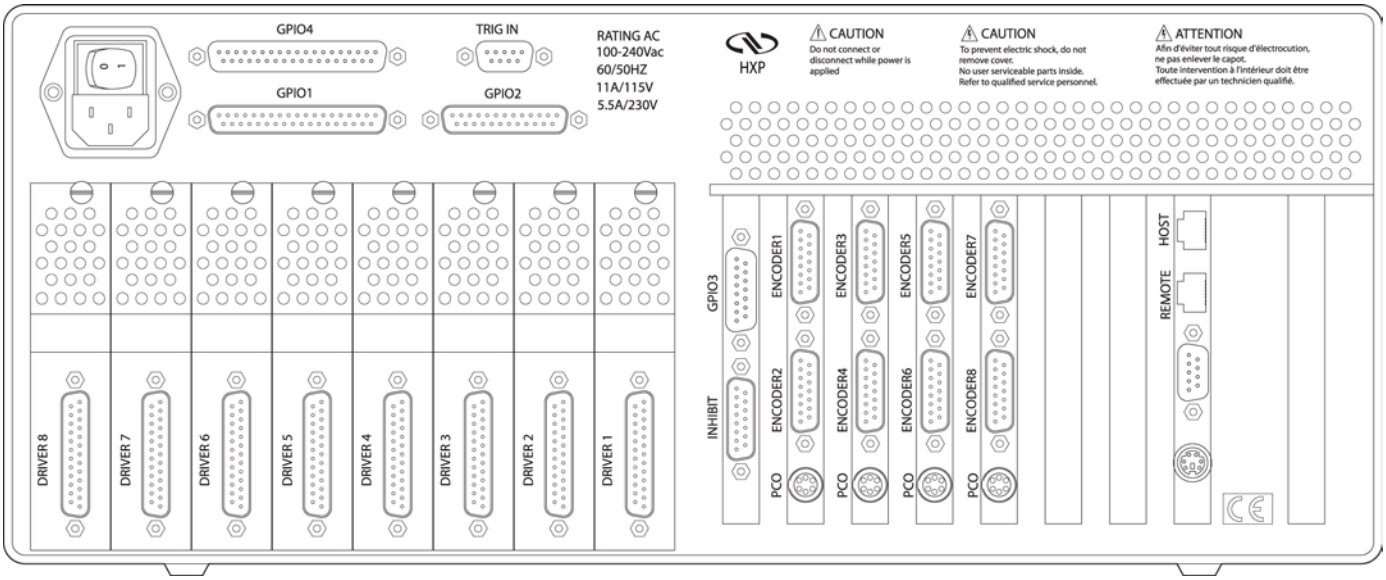


Weight:	16 kg (32 lb)
Input voltage:	100–240 VAC
Input current:	11 A/115 V 5.5 A/230 V
Frequency:	60/50 Hz

14.2 Rear Panel Connectors



Hardware Configuration for Hexapod only.



Hardware Configuration for Hexapod and upto 2 Single Axes.

14.3 Environmental Requirements

Temperature range:

Storage: -20 to +80 °C

Operating: +5 to +35 °C

Relative Humidity (Non-condensing):

Storage: 10 to 95% RH

Operating: 10 to 85% RH

Altitude:

Storage: To 10,000 ft (3000 m)

Operating: To 5,000 ft (1500 m)

15.0 Appendix B: General I/O Description

This paragraph briefly describes all HXP signal types.

Description of each HXP connector interface is detailed in further paragraphs.

15.1 Digital I/Os (All GPIO, Inhibit and Trigger In and PCO Connectors)

All digital I/Os are TTL compatible:

- All digital I/Os are not isolated, but are referenced to electrical ground (GND).
- Input levels must be between 0 V and +5 V.
- Output levels should be at least +5 V (up to 30 V absolute maximum rating with open collector outputs).
- Outputs must be pulled up to the user external power supply (+5 V to +24 V). This external power supply must be referenced to the HXP ground (GND).

All digital I/Os are refreshed asynchronously on user requests. Therefore, digital inputs or outputs have no refreshment rate.

Typical availability delay is 100µs due to function treatment.

All digital inputs are identical, except for GPIO3 inhibition input (described with GPIO3).

All digital inputs are in negative logic and have internal +5 V pull up resistors.

15.1.1 Digital Inputs

Parameter	Symbol	Min.	Max.	Units
Low Level Input Voltage	V_{IL}	0	0.8	V
High Level Input Voltage	V_{IH}	1.6	5	V
Input Current LOW	I_{IL}	–	-2.5	mA
Input Current HIGH	I_{IH}	–	0.4	mA

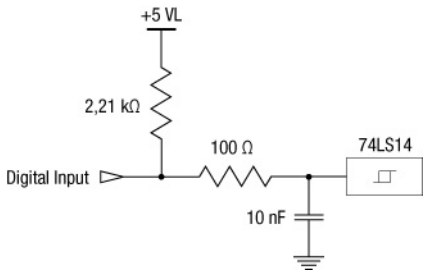


Figure 38: Digital TTL Input.

GPIOn inputs (n = 1 to 4) can be accessed via the GPIODigitalGet(GPIOn.DI, …) function.

All digital outputs are identical.

All digital outputs are in negative logic (NPN open collector, 74LS06 TTL type circuit) and have no internal pull up to permit levels above +5 V.

15.1.2 Digital Outputs

Parameter	Symbol	Min.	Max.	Units
Low Level Output Voltage	V_{OL}	0	1	V
High Level Output Voltage	V_{OH}	2.4	30	V
Input Current LOW	I_{OL}	–	-40	mA
Input Current HIGH	I_{OH}	–	0.2	mA

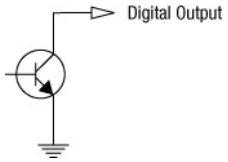


Figure 39: Open Collector Digital Output.

GPIOn outputs (n = 1 to 4) can be accessed via the GPIODigitalSet(GPIOn.DO, …) function.

15.2 Digital Encoder Inputs (Driver Boards & DRV00)

All digital encoder inputs are RS-422 standard compliant:

- All digital encoder signals are not isolated, but are referenced to the electrical ground (GND).
- Encoder signals must be differential pairs (using 26LS31 or MC3487 line driver type circuits). Encoder inputs have a terminating impedance of 120 Ω.
- Inputs are always routed on differential pairs. For a high level of signal integrity, we recommend using shielded twisted pairs of wires for each differential signal.
- •Encoder power supply is +5 V @ 250 mA maximum (referenced to the electrical ground) and is sourced directly by the driver boards.

15.3 Digital Servitudes (Driver Boards, DRV00 & Analog Encoders Connectors)

All servitude inputs are TTL compatible:

- All servitude inputs are not isolated, but are referenced to the electrical ground (GND).
- Input levels must be between 0 V and +5 V.

All servitude inputs are refreshed synchronously with control loop (10 kHz).

All servitude inputs are identical.

All servitude inputs expect normally closed sensors referenced to ground (input is activated if the sensor is open) and have internal 2.2 KΩ pull up resistors to the +5 V.

15.4 Analog Encoder Inputs (Analog Encoder Connectors)

Analog encoder interface comply with the Heidenhain LIF481 glass scales wiring standard.

15.5 Analog I/O (GPIO2 Connector)

15.5.1 Analog Inputs

The 4 analog inputs have ±10 V range, 14 Bit resolution, and a 15 kHz 2nd order low pass filter front end.

In all cases, the analog input values must be within the ±10 V. The analog input impedance is typically 22 kΩ. The maximum input current is ±500 µA.

1 LSB = 20 V/16384 ≈ 1.22 mV

The maximum offset error is ±17,1 mV.

15.5.2 Analog Outputs

The 4 analog outputs have ±10 V range and 16 Bit resolution. The maximum offset error is ±2 mV, and the maximum gain error is ±6 LSB. The output settling time is typically 50 µsec at 1% of the target value (output filter is a 15 kHz 1st order low pass filter).

Analog outputs are voltage outputs (output current less than 1 mA), so to use them properly, they must be connected to impedance higher than 10 kΩ.

1 LSB =: 20 V/65536 ≈ 0.3 mV.

Analog outputs can be accessed via the GPIOAnalogSet(GPIO2.DACn,…) function.

16.0 Appendix C: Power Inhibit Connector

PIN	Signal type	Description	PIN	Signal type	Description
1	Supply	GND	9	Reserved	
2	Inhibition	Must be connected to GND	10	Reserved	
3	Reserved		11	Reserved	
4	Reserved		12	Reserved	
5	Reserved		13	Reserved	
6	Reserved		14	Reserved	
7	Reserved		15	Reserved	
8	Reserved				

Figure 40: Inhibition connector.

This connector is provided for the wiring of a remote STOP ALL switch.

It has the same effect as the front panel STOP ALL button.

Inhibition input is a standard TTL input.

Inhibition (Pin #2), must always be connected to GND during normal controller operation.

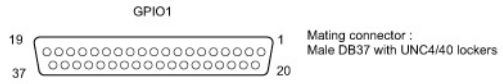
An open circuit is equivalent to pressing STOP ALL on the front panel. Wire the switch contacts normally closed.

NOTE

Connecting more than one switch is not recommended on this input.

17.0 Appendix D: GPIO Connectors

17.1 GPIO1 Connector



PIN	Signal type	Description	PIN	Signal type	Function
1	N.C.		20	Supply	GND
2	Supply	+12V supply output<100mA	21	Supply	GND
3	Supply	+5V supply output<100mA	22	Supply	GND
4	TTL input	Input 1	23	Supply	GND
5	TTL input	Input 2	24	Supply	GND
6	TTL input	Input 3	25	Supply	GND
7	TTL input	Input 4	26	Supply	GND
8	TTL input	Input 5	27	Supply	GND
9	TTL input	Input 6	28	Supply	GND
10	TTL input	Input 7	29	Supply	GND
11	TTL input	Input 8	30	Supply	GND
12	O.C. output	Output 1	31	Supply	GND
13	O.C. output	Output 2	32	Supply	GND
14	O.C. output	Output 3	33	Supply	GND
15	O.C. output	Output 4	34	Supply	GND
16	O.C. output	Output 5	35	Supply	GND
17	O.C. output	Output 6	36	Supply	GND
18	O.C. output	Output 7	37	Supply	GND
19	O.C. output	Output 8			

Figure 41: GPIO1 Digital I/O Connector.

General Purpose Inputs Outputs GPIO1 is the main HXP digital I/O connector.

17.2 GPIO2 Connector

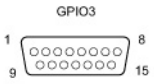
PIN	Signal type	Description	PIN	Signal type	Description
1	Supply	GND	14	Analog input	Analog input 1
2	N.C.		15	Analog input	Analog input 2
3	TTL input	Input 1	16	Analog input	Analog input 3
4	TTL input	Input 2	17	Analog input	Analog input 4
5	TTL input	Input 3	18	Supply	GND
6	TTL input	Input 4	19	Analog output	Analog output 1
7	TTL input	Input 5	20	Analog output	Analog output 2
8	N.C.		21	Analog output	Analog output 3
9	N.C.		22	Analog output	Analog output 4
10	Reserved		23	Supply	GND
11	Reserved		24	TTL input	Input 6
12	Reserved		25	Supply	GND
13	Supply	GND			

Figure 42: GPIO2 Analog & Digital Connector.

General Purpose Inputs Outputs GPIO2 is an additional digital input connector.

This connector is also the main analog I/O connector with 4 analog inputs and 4 analog outputs.

17.3 GPIO3 Connector



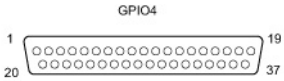
Mating connector :
Female DB15 with UNC4/40 lockers

PIN	Signal type	Function	PIN	Signal type	Function
1	Supply	GND	9	TTL input	Input 1
2	Reserved		10	TTL input	Input 2
3	O.C. output	Output 1	11	TTL input	Input 3
4	O.C. output	Output 2	12	TTL input	Input 4
5	O.C. output	Output 3	13	TTL input	Input 5
6	O.C. output	Output 4	14	TTL input	Input 6
7	O.C. output	Output 5	15	Supply	+5V supply output<100mA
8	O.C. output	Output 6			

Figure 43: GPIO3 Digital I/O Connector.

General Purpose Inputs Outputs GPIO3 is a digital I/O connector.

17.4 GPIO4 Connector



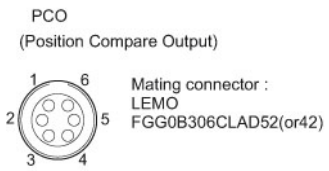
Mating connector :
Female DB37 with UNC4/40 lockers

PIN	Signal type	Description	PIN	Signal type	Description
1	TTL input	Input 1	20	TTL input	Input 2
2	TTL input	Input 3	21	TTL input	Input 4
3	TTL input	Input 5	22	TTL input	Input 6
4	TTL input	Input 7	23	TTL input	Input 8
5	TTL input	Input 9	24	TTL input	Input 10
6	TTL input	Input 11	25	TTL input	Input 12
7	Supply	GND	26	Supply	+5V supply output <100mA
8	Supply	GND	27	O.C. output	Output 1
9	O.C. output	Output 2	28	O.C. output	Output 3
10	O.C. output	Output 4	29	O.C. output	Output 5
11	O.C. output	Output 6	30	Supply	+5V supply output <100mA
12	Supply	GND	31	TTL input	Input 13
13	TTL input	Input 14	32	TTL input	Input 15
14	TTL input	Input 16	33	O.C. output	Output 7
15	O.C. output	Output 8	34	O.C. output	Output 9
16	O.C. output	Output 10	35	O.C. output	Output 11
17	O.C. output	Output 12	36	O.C. output	Output 13
18	O.C. output	Output 14	37	O.C. output	Output 15
19	O.C. output	Output 16			

Figure 44: GPIO4 Additional Digital I/O Connector.

General Purpose Inputs Outputs GPIO4 is an additional digital I/O connector.

18.0 Appendix E: PCO Connector



PIN	Distance spaced pulses or Tim spaced pulses	AquadB signals
1	+5 VL	+5 VL
2	Axis 1: Pulse enable	Axis 1: A
3	Axis 1: Pulse	Axis 1: B
4	Axis 2: Pulse Enable	Axis 2: A
5	Axis 2: Pulse	Axis 2: B
6	GND	GND

Figure 45: Position Compare Output Connector.

There is one PCO connector for every two axes. Axis #1 refers to the upper (odd) encoder plug and axis #2 refers to the lower (even) encoder plug. The signals provided on this plug depend on the configuration of the output triggers, see section 13, Output trigger, for more details.

The state of the enable signal is low when the stage is inside the programmed position compare window.

Note also, that only the falling edge of the trigger pulse is precise and only this edge should be used for synchronization irrespactable from the PCOPulseWidth setting.

The duration of the pulse is 200 nsec by default and can be modified using the function **PositionerPositionComparePulseParametersSet()**. Possible values for the PCOPulseWidth are: 0.2 (default), 1, 2.5 and 10 (µs). Successive trigger pulses should have a minimum time lag equivalent to the PCOPulseWidth time times two.

The signals are open collector type and accept up to 30 Volts and 40 mA

The +5V output provided on the PCO connector can be used to pull-up these outputs and can supply 50 mA max.

NOTE

To ensure fast transitions with an open collector, it is necessary to have enough current to speed-up the transistor's junction capacitor charge / discharge. A good value is around 10 mA. So to pull-up the PCO signals to +5 V a 470 Ω resistor can be used.

Refer to section B.1 Digital I/Os, § Digital Outputs for detailed electrical description.

19.0 Appendix F: Motor Driver Cards

19.1 DC and Stepper Motor Driver XPS-DRV01

PIN	Function		PIN	Function	
	D.C. MOTOR	STEPPER MOTOR		D.C. MOTOR	STEPPER MOTOR
1	Tachometer+	Ph 1	15	Index	Index
2	Tachometer+	Ph 1	16	Limit ground	Limit ground
3	Tachometer-	Ph 2	17	+Travel limit	+Travel limit
4	Tachometer-	Ph 2	18	-Travel limit	-Travel limit
5	Motor+	Ph 3	19	Encoder A	Encoder A
6	Motor+	Ph 3	20	Encoder B	Encoder B
7	Motor-	Ph 4	21	+5V	+5V
8	Motor-	Ph 4	22	Encoder ground	Encoder ground
9	Reserved	Common 3&4	23	Encoder /A	Encoder /A
10	Reserved	Common 3&4	24	Encoder /B	Encoder /B
11	Reserved	Common 1&2	25	/index	/index
12	Reserved	Common 1&2			
13	Origin	Origin			
14	Shld GND	Shld GND			

Figure 46:XPS-DRV01 Motor Driver Connectors.

Motor + This output must be connected to the positive lead of the DC motor. The voltage seen at this pin is pulse-width modulated with maximum amplitude of 48 V DC.

Motor - This output must be connected to the negative lead of the DC motor. The voltage seen at this pin is pulse-width modulated with maximum amplitude of 48 V DC.

Ph1 This output must be connected to Winding A+ lead of a two-phase stepper motor. The voltage seen at this pin is pulse-width modulated with maximum amplitude of 48 V DC.

Ph2 This output must be connected to Winding A- lead of a two-phase stepper motor. The voltage seen at this pin is pulse-width modulated with maximum amplitude of 48 V DC.

Ph3 This output must be connected to Winding B+ lead of a two-phase stepper motor. The voltage seen at this pin is pulse-width modulated with maximum amplitude of 48 V DC.

Ph4 This output must be connected to Winding B- lead of a two-phase stepper motor. The voltage seen at this pin is pulse-width modulated with maximum amplitude of 48 V DC.

Common 3&4 This output must be connected to the center tab of Winding B of a two-phase stepper motor. The voltage seen at this pin is pulse-width modulated with maximum amplitude of 48 V DC.

Common 1&2 This output must be connected to the center tab of Winding A of a two-phase stepper motor. The voltage seen at this pin is pulse-width modulated with maximum amplitude of 48 V DC.

+ Travel limit This input is pulled-up to +5 V with a 2.2 kΩ resistor by the controller and represents the stage positive direction hardware travel limit.

- Travel limit This input is pulled-up to +5 V with a 2.2 kΩ resistor by the controller and represents the stage negative direction hardware travel limit.

Encoder A & /A These A and /A inputs are differential inputs. Signals are compliant with RS422 electrical standard and are received with a 26LS32 differential line receiver. A resistor of 120 Ω adapts the input impedance. The A and /A encoder signals originate from the stage position feedback circuitry and are used for position tracking.

Encoder B and /B These B and /B inputs are differential inputs. Signals are compliant with RS-422 electrical standard and are received with a 26LS32 differential line receiver. A resistor of 120 Ω adapts the input impedance. The B and /B encoder signals originate from the stage position feedback circuitry and are used for position tracking.

Index & /Index These Index and /Index inputs are differential inputs. Signals are compliant with RS422 electrical standard and are received with a 26LS32 differential line receiver. A resistor of 120 Ω adapts the input impedance. The Index and /Index signals originate from the stage and are used for homing the stage to a repeatable location.

Encoder ground Ground reference for encoder feedback.

Origin This input is pulled-up to +5 V with a 2.2 kΩ resistor by the controller. The Origin signal originates from the stage and is used for homing the stage to a repeatable location.

+5 V (DRV01: 250 mA Maximum) A +5 V DC supply is available from the driver. This supply is provided for stage home, index, travel limit, and encoder feedback circuitry.

Limit ground Ground for stage travel limit signals. Limit ground is combined with digital ground at the controller side.

Shield GND Motor cable shield ground.

Tachometer + & Tachometer – These inputs are used to receive tachometer voltage information. This voltage depends on the output voltage rating of the employed tachometer.

19.2 Three Phases AC Brushless Driver XPS-DRV02

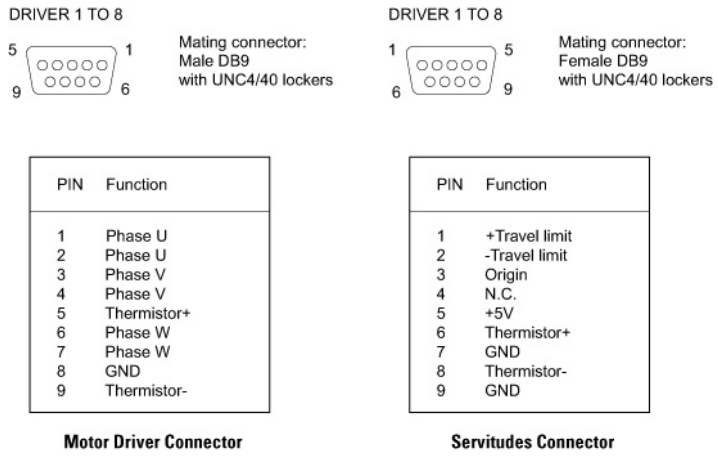
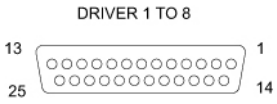


Figure 47: XPS-DRV02 Motor Driver Connectors.

19.3 DC Motor Driver XPS-DRV03



PIN	Function	PIN	Function
1	Tachometer+	15	Index
2	Tachometer+	16	Limit ground
3	Tachometer-	17	+Travel limit
4	Tachometer-	18	-Travel limit
5	Motor+	19	Encoder A
6	Motor+	20	Encoder B
7	Motor-	21	+5V
8	Motor-	22	Encoder ground
9	N.C.	23	Encoder /A
10	N.C.	24	Encoder /B
11	N.C.	25	/index
12	N.C.		
13	Origin		
14	Shld GND		

Figure 48: XPS-DRV03 Motor Driver Connectors.

19.4 Pass-Through Board Connector (25-Pin D-Sub) XPS-DRV00

WARNING

This pass-through board connector takes the place of the motor interface connector only if this axis is connected to an external motor driver.

PIN	Function	PIN	Function
1	Reserved	14	Reserved
2	+5 V	15	Inhibition output
3	Origin input	16	Reserved
4	– Travel limit input	17	Reserved
5	+ Travel limit input	18	Reserved
6	Main fault input	19	Encoder /A input
7	Encoder A input	20	Encoder /B input
8	Encoder B input	21	/Index input
9	Index input	22	Reserved
10	Pulse / Pulse+ output	23	GND
11	Direction / Pulse- output	24	N.C.
12	Analog A output	25	GND
13	Analog B output		

Figure 49: DRV00 Pass-Through Connector.

Analog A output and Analog B output have 16 bits resolution and are ±10 V output. These signals are used to command an external driver.

20.0 Appendix G: Analog Encoder Connector

PIN	Function	PIN	Function
1	Analog VA (0.5Vpp)	9	Analog /VA (0.5Vpp)
2	0V	10	0V
3	Analog VB (0.5Vpp)	11	Analog /VB (0.5Vpp)
4	+5VA	12	+5VL
5	N.C.	13	N.C.
6	Limit (TTL input)	14	Analog VI (0.5Vpp)
7	Analog /VI (0.5Vpp)	15	N.C.
8	Home (TTL input)		

Figure 50: Analog Encoders Connector.

This connector is used to receive sine/cosine encoder signals.

The sinusoidal position signals, sine and cosine, must be phase-shifted by 90° and have signal levels of approximately 1 Vpp. Each of these two signals is composed of an analog sinusoidal signal and his complement entering in a differential amplifier (Sine = Analog VA - Analog /VA).

Analog VA, Analog /VA, Analog VB, Analog /VB, Analog VI and Analog /VI:

Levels for these signals must be 0.5 Vpp.

VA, /VA, VB and /VB inputs are the sine and cosine signals from the encoder glass scale.

VI and /VI inputs are used to receive Index information from the encoder glass scale.

+5 VA:

This +5 V DC supply is provided for supplying the encoder.

+5 VL:

This +5 V DC supply is provided for supplying digital circuits (Limit and Home).

Limit and Home are TTL inputs for Limit switch management and homing purposes directly from the encoder glass scale.

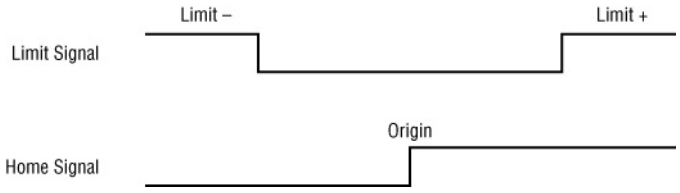
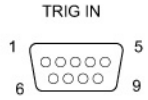


Figure 51: Heidenhain Servitude TTL Input Signals.

21.0 Appendix H: Trigger IN Connector



Mating connector :
Female DB9 with UNC4/40 lockers

PIN	Function	PIN	Function
1	GND	6	Syncro input
2	GND	7	N.C.
3	N.C.	8	N.C.
4	N.C.	9	N.C.
5	N.C.		

Figure 52: Trigger Input Connector.

Synchro is a TTL input. It is used to trig the HXP controller acquisition (External gathering).

A low to high transition will latch all encoders and analog inputs inside the controller.

Service Form

Your Local Representative

Tel.: _____

Fax: _____

Name: _____

Company: _____

Address: _____

Country: _____

P.O. Number: _____

Item(s) Being Returned: _____

Model#: _____

Return authorization #: _____
(Please obtain prior to return of item)

Date: _____

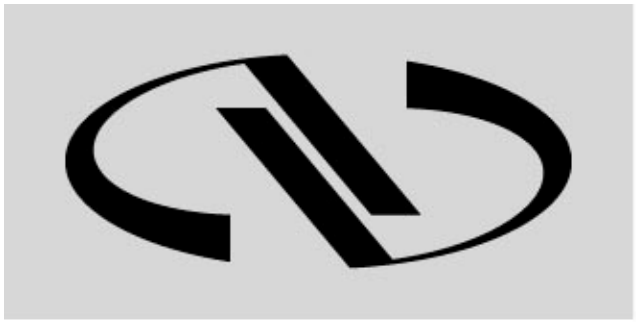
Phone Number: _____

Fax Number: _____

Serial #: _____

Description: _____

Reasons of return of goods (please list any specific problems): _____



Newport®

Experience | Solutions

Visit Newport Online at:
www.newport.com

North America & Asia
Newport Corporation
1791 Deere Ave.
Irvine, CA 92606, USA

Sales
Tel.: (800) 222-6440
e-mail: sales@newport.com

Technical Support
Tel.: (800) 222-6440
e-mail: tech@newport.com

Service, RMAs & Returns
Tel.: (800) 222-6440
e-mail: service@newport.com

Europe
MICRO-CONTROLE Spectra-Physics S.A.S
9, rue du Bois Sauvage
91055 Évry CEDEX
France

Sales
Tel.: +33 (0)1.60.91.68.68
e-mail: france@newport.com

Technical Support
e-mail: tech_europe@newport.com

Service & Returns
Tel.: +33 (0)2.38.40.51.55